République du Sénégal

Université Cheikh Anta Diop de Dakar

Ecole Supérieure Polytechnique

Centre de Dakar

Département Génie Informatique

Mémoire

Pour l'obtention du

Diplôme d'ingénieur technologue (DIT)

Modélisation générique de systèmes complexes : Edition des modèles MIMOSA

Présenté par **Mohamed Bécaye Touré**

Encadreurs :

M. Jean Pierre Müller
M. Alassane Bah
M. Grégoire Leclerc

Janvier 2006

Avant-propos

Avant d'entamer cet exposé je tiens à remercier l'Ecole Supérieure Polytechnique qui m'a accueilli, tout le long de ma formation, ainsi que les enseignants du département informatique qui m'ont tous prodigués de merveilleux conseils et plus particulièrement M. Alassane Bah et M. Mamadou Niang qui m'ont encadré lors de ce travail.

Mes remerciements vont aussi au CIRAD qui m'a accueilli lors de mon stage et plus particulièrement au Pr. Jean Pierre Mullër et M. Grégoire Leclerc. Je les remercie pour leurs conseils et leur encadrement.

J'adresse mes remerciements les plus sincères à l'ensemble des membres de ma famille, mes parents qui m'ont aidé, soutenu et sans qui d'ailleurs rien ne serait possible. Je les remercie et leurs dédie ce travail.

Table des matières

Avant-propos *2*
Table des matières *3*
Table des illustrations *5*
Introduction *6*
Chapitre I. Modélisation et systèmes multi-agents *8*
A. La modélisation 8
B. Le modèle 9
C. La modélisation informatique 9
D. Les systèmes multi-agents 12
1. Qu'est ce qu'un agent ? 12
2. Les systèmes multi-agents 13
3. Pourquoi parler des systèmes multi-agents ? 13
E. Les plateformes de modélisation et simulation existantes 13
F. La plateforme MIMOSA 15
Chapitre II. Les outils *17*
A. La programmation orientée objet (POO) 17
B. Le langage Java 17
C. Le formalisme UML 17
1. Le diagramme de classes 18
a) Les classes 18
b) Les associations 19
c) Les agrégations 19
d) Les compositions 19
e) La généralisation 20
2. Le diagramme de cas d'utilisation 21
a) Les acteurs 21
b) Les cas d'utilisation 21
c) La relation d'inclusion 21
3. Le diagramme de séquence 22
a) Les interactions 22
b) Les activations et envois de messages 22
D. Le format XML 24
1. Le fichier XML 24
2. La DTD 25
3. Les API Java pour XML 25
4. L'API SAX 25
5. L' API DOM 25
6. L'avantage du SAX 26
Chapitre III. Dossier technique *27*

A. La notion d'éditeur de modèle 27

B. Ce qu'il s'agit de faire 29

C. Généricité, boites à outils, formalismes 33

1. Concepts de modèles, abstractions et définitions 33
   a) Notions structurelles et notions dynamiques 33
   b) La boite à outils MIMOSA 33
   c) La boite à outils S-Edit 35
   d) Résultats des jeux : même boite à outils ? 36
2. Implémentations 38
   a) Implémentation de la boite à outils MIMOSA 38
   b) Implémentation de la boite à outils S-Edit 43
3. Sauvegarde des modèles 45
   a) Sauvegarde des modèles MIMOSA 46
   b) Sauvegarde des modèles S-Edit 46
   c) Comparaison 46
4. Bilan 47

Chapitre IV. Edition de modèles MIMOSA 48

A. Assimilations 48

B. Assimilation par héritage et interfaçage 49

1. Description 49
2. Implémentation 49
3. Dépendances 51
4. Protocole de description : création de catégories S-Edit 55
5. Sauvegarde XML 60
6. Bilan de l'assimilation par héritage 61

C. L'assimilation par délégation 63

1. Description 63
2. Implémentation 63
3. Dépendances 66
4. Bilan de l'assimilation par délégation 66

Chapitre V. Conclusion et perspectives 68

A. Résumé 68

B. Fonctionnalités supplémentaires 68

C. Développement de la plateforme 69

D. Conclusion 70

Annexe A. Bibliographie 71

Annexe B. DTD des fichiers XML de description de formalismes S-Edit 73

Annexe C. Guide utilisateur de l'éditeur de modèles MIMOSA 75

Index 76

Table des illustrations

Figure 1. Représentation graphique des classes 18
Figure 2. Représentation graphique d'une association 19
Figure 3. Représentation graphique d'une agrégation 19
Figure 4. Représentation graphique de la composition 20
Figure 5. Représentation graphique de la relation de généralisation entre classes 20
Figure 6. Représentation graphique des cas d'utilisation 21
Figure 7. Représentation graphique de la relation d'inclusion 22
Figure 8. Un digramme de séquence 23
Figure 9. Edition d'un modèle de population 28
Figure 10. Edition d'un automate cellulaire 29
Figure 11. Fonctionnalités de la plateforme MIMOSA 30
Figure 12. Fonctionnalités du système S-Edit 31
Figure 13. Modélisation de la plateforme MIMOSA selon le pattern MVC 33
Figure 14. Les notions structurelles de la boite à outil MIMOSA 35
Figure 15. Légère spécialisation de S-Edit: connotation graphique 37
Figure 16. La notion d'état : degrés d'abstraction différents 38
Figure 17. Comparaison action, événements et fonctions de transition 38
Figure 18. La boite à outils de la plateforme MIMOSA 42
Figure 19. La boite à outils S-Edit 45
Figure 20. Description interactive et statique des catégories d'objets 47
Figure 21. Une boite à outils unifiée 51
Figure 22.Contrôleur (MVC) 55
Figure 23. Editeurs 56
Figure 24. Editeur des propriétés graphiques 57
Figure 25. Description de la catégorie de graphe 58
Figure 26. Sélection des noeuds d'un graphe 58
Figure 27. Le protocole de description des catégories 59
Figure 28. Gestion de la sauvegarde à travers l'API SAX 60
Figure 29. Boites à outils : assimilation par délégation 65

Introduction

Le diplôme d'Ingénieur technologue (DIT) option informatique est un diplôme d'ingénieur délivré par le Département Génie informatique de l'Ecole supérieure polytechnique (ESP) de Dakar. A la fin du cycle de formation qualifiant à ce titre, il est requis des étudiants qu'ils présentent un exposé sur un sujet leur permettant de mettre en œuvre les connaissances et le savoir faire acquis dans leur formation.

C'est dans cette optique, que j'ai effectué un stage de six mois au Laboratoire Informatique et Réseaux Télécom (LIRT) de l'ESP, en partenariat avec le Centre de coopération internationale en recherche agronomique pour le développement (CIRAD). Le sujet traité lors de ce stage porte sur la conception d'un éditeur de modèle pour une plateforme générique de modélisation et de simulation multi-agents, la plateforme MIMOSA.

La plateforme MIMOSA a été mise en place dans le cadre d'un projet du même nom, le projet MIMOSA (Méthodes Informatiques Modélisation et Simulations Agents), visant à fédérer les travaux des équipes de recherche en simulation multi-agents.

La communauté francophone de recherche en simulation multi-agents s'est structurée et distinguée par des travaux originaux sur la gestion de l'environnement et des territoires. Un réseau de compétences multidisciplinaires s'est constitué, supporté par un important travail informatique, permettant ainsi de développer les infrastructures nécessaires. La plateforme MIMOSA se veut une synthèse du meilleur des expériences et de l'existant technique.

Au regard de l'aspect collaboratif du projet, il nous a été demandé d'utiliser le système d'édition de diagrammes de formalisme quelconque, le système S-Edit mis en place par le LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier). Notre tâche a alors été de déterminer dans quelle mesure nous pouvions lier le système S-Edit à la plateforme MIMOSA.

Ce mémoire tente d'apporter des éléments de réponse, voire une solution au problème identifié. Pour appréhender la question, il convient d'avoir une idée précise du domaine d'utilisation de la plateforme à savoir la modélisation et la simulation. Partant, la première partie de ce document porte sur l'état des lieux dans le domaine de la modélisation et l'apport de l'informatique à cette discipline, notamment avec les techniques multi-agents. Nous pourrons ensuite situer le projet MIMOSA dans son contexte. Dans ce premier chapitre nous traiterons des objectifs du projet ainsi que des principes de base sur lesquelles s'appuie sa démarche.

Le contexte de ce travail est défini dans la première partie de l'exposé, avant d'aborder la faisabilité du couplage. Pour ce faire, sera dressé, à des fins comparatives, un dossier technique sur la plateforme MIMOSA et sur le système S-Edit. Dans ce chapitre les systèmes étudiés seront modélisés selon certains formalismes, notamment le formalisme UML . A travers ces formalismes certains concepts seront visualisés, à savoir les objets en programmation. Nous évoquerons des technologies, des langages, tel le langage Java, mettant en œuvre ces concepts. Afin que ces parties soient compréhensibles à un groupe plus vaste que la seule communauté des informaticiens, nous réserverons le deuxième chapitre de ce document à une présentation de ces outils.

Une fois les systèmes étudiés, nous proposerons dans le quatrième chapitre une réponse ou plutôt une solution. Avant de conclure nous ferons le point sur l'implémentation de la solution préconisée, les difficultés rencontrées, tout en mettant en exergue les défauts que nous avons cru déceler, tant sur MIMOSA que sur S-Edit. Nous émettrons des suggestions, tout en essayant d'évaluer l'apport de ce travail au projet. Quelques avis critiques seront formulés et des indications données quant aux perspectives pour l'amélioration de ce travail tant au niveau conceptuel qu'au niveau de l'implémentation. Un guide d'utilisation de l'éditeur développé est présenté en annexe.

Chapitre I.
Modélisation et systèmes multi-agents

A.La modélisation

L'activité scientifique produit des connaissances. Avec la connaissance, l'être humain élabore des procédés pour résoudre les problèmes auquel il est confronté. Disons le, c'est par la connaissance que l'homme met au point des vaccins et des médicaments, qu'il innove, améliore son cadre de vie et son confort de vie, crée des avions, communique, met en place des réseaux, met en place des stratégies économiques, etc. C'est aussi par la connaissance que sont inventées, malheureusement, des armes de destruction, outils de guerre, et les redoutables bombes atomiques. Au cours du siècle dernier et en ce début de millénaire, la science a permis une amélioration du niveau de vie des habitants du globe, avec des avancées sans précédents au niveau de la médecine et des innovations technologique. Mais elle aura aussi contribué à une certaine détérioration de ce niveau de vie en provoquant des catastrophes humanitaires sans précédent lors des nombreux conflits qui ont ponctué l'histoire contemporaine. La connaissance scientifique occupe donc une position centrale dans nos sociétés ; elle constitue un formidable outil de développement pour les Etats et représente un enjeu stratégique dans l'exercice du pouvoir.

A la base d'une connaissance scientifique il y a un acteur : le savant, le chercheur. La connaissance et le résultat scientifique, sont produits par le chercheur au terme d'un processus qualifié de démarche ou méthode scientifique. Cette démarche comporte cinq étapes :

- l'observation
- la formulation d'hypothèse
- la vérification de résultats
- l'élaboration de théories
- la communication.

Avant de débuter l'observation, le chercheur limite son champ d'observation sur l'entité qu'il prévoit d'étudier. Toutes les propriétés de l'entité ne l'intéressent pas. Il a une idée précise de ce qu'il veut observer. Dans son esprit, à cet instant l'entité se limite à la conception qu'il se fait d'elle. C'est le début de la modélisation : la réalité est simplifiée, c'est l'élaboration d'une idée, d'une vision de la réalité, d'une conception de la réalité, c'est l'élaboration d'un modèle. Après observation, le chercheur émet des hypothèses, il a acquis des précisions hypothétiques sur les concepts de son modèle. Il va vouloir valider, vérifier son nouveau modèle. Il passe à une seconde observation, plus restreinte, plus formalisée : c'est l'expérimentation. Au terme de l'expérimentation il conserve ou rejette les précisions, les aspects, les attributs des concepts du modèle. Il finalise le formalisme de son modèle.

B.Le modèle

Le Robert donne plusieurs définitions du mot modèle dont celle-ci, dans le domaine de la science : « représentation simplifiée d'un processus, d'un système » Un modèle est un point de vue sur le monde, une simplification de la réalité, une conception du monde. L'être humain a une vision morcelée du monde, il identifie des entités. Aussi Maturana [2] considère-t-il le modèle comme étant un ensemble de concepts, un discours qui articule des concepts. Etant une simplification, tout modèle a ses limites, son domaine de validité. Le modèle n'est pas la réalité : en dehors de ses limite, ses résultats ne collent plus à la réalité ; ils ne traduisent alors que les propriétés du modèle.

C.La modélisation informatique

Un modèle informatique ou modèle numérique est l'implémentation d'un modèle sur un ordinateur. Quelle est l'utilité d'implémenter un modèle sur ordinateur ? Un modèle est un point de vue sur le monde, un monde soumis à une certaine temporalité. Un bon modèle doit rendre compte de l'évolution des concepts dans le temps, il doit être prédictif. Un modèle prédictif peut être perçu comme une fonction mathématique variant au cours du temps, l'ordinateur étant le calculateur permettant de visualiser les variations de cette fonction. Le processus de variation des états du modèle est appelé simulation. Un modèle étant une perception plus ou moins exacte de la réalité, la simulation permet de prévoir avec plus ou moins d'exactitude l'avenir de cette réalité. Pourquoi prévoir ou plutôt qui a besoin de prévoir?

Deux acteurs : le chercheur et le décideur. Le premier, le chercheur, nous l'avons rencontré dès notre introduction à la modélisation. C'est lui qui établit le modèle et lui confère sa

prédictibilité. Après que preuve a été faite de l'efficience de son modèle, celui-ci est utilisé dans des simulations à titre d'expérimentation virtuelle. Ce genre d'expérimentation ne se substitue cependant pas à l'expérimentation classique, mais les expériences étant fort coûteuses, il permet de sélectionner l'expérience adéquate parmi plusieurs. Au niveau du deuxième acteur, le décideur, la simulation informatique est utilisée à des mêmes fins d'aide à la décision. En effet « gouverner c'est prévoir », la gestion de l'environnement et des territoires nécessite des prévisions sur ce qu'il convient d'appeler un éco-sociosystème, elle nécessite donc l'établissement de plusieurs modèles en interaction rendant compte de la réalité d'un éco-sociosystème.

Une politique de reboisement doit prendre en compte aussi bien les dynamiques écologiques de la région cible, la biologie des essences végétales à planter, que le comportement des bûcherons. Pour cela les chercheurs en biologie végétale, en sciences écologiques et en sciences sociales devront s'associer à l'informaticien pour mettre en place un modèle écologique et un modèle de société. Mais quels sont les outils d'on dispose l'informaticien pour implémenter et simuler les modèles de ses collègues ?

L'informaticien dispose d'une machine ; sa machine permet de stocker des informations et d'effectuer des traitements sur cette information. Ses collègues lui disent : « Voila, nous nous sommes des modélisateurs, nous concevons des modèles, nos modèles sont des ensembles de concepts, les concepts sont caractérisé par un état, l'état est un ensemble d'information, à un temps **t** une fonction va modifier l'état et donc les informations caractéristiques de l'état seront modifiées ; nous nous sommes d'excellents mathématiciens nous savons calculer la transition. Mais nous voulons étudier les variations d'état sur des milliards d'années. Or nous ne sommes qu'humains, nous sommes trop lents et nous allons nous fatiguer. Ce que nous voulons c'est que tu automatises ce calcul, pour nous de sorte que quand nous aurons fini notre café nous sachions comment va évoluer l'univers ». L'informaticien sait très bien que l'écologiste et le sociologue n'ont fait appel à lui que parce qu'ils ne savaient pas utiliser la machine permettant de simuler l'évolution de leurs modèles. Celui-ci génial utilisateur leur dits : « Je vais utiliser les capacités de stockage de ma machine pour stocker l'état des concepts, et j'utiliserai les capacités de traitement pour modéliser votre fonction. Car vous n'êtes pas les seuls à modéliser. Moi aussi je modélise et ma tache et plus difficile que la votre, parce qu'il est des jours où je me mets à la place de ma machine je modélise pour elle vos informations et vos traitements de sorte qu'elle puisse comprendre les humains. Vous n'avez pas idée comme ma tache est difficile, quand je me mets à la place de ma machine je ne vois que des zéros et uns et je dois percevoir ce que vous me demandez en termes de zéros et de uns. J'effectue le transfert de point de vue, du votre vers celui de ma machine. Il faut noter que le jour où ma machine sera capable d'effectuer toute seule cette tache, ce jour

là elle sera devenue intelligente et je n'aurais plus de métier. Mais pour l'heure c'est de loin ma tache la plus difficile. Et comme je suis aussi humain et que cette tache est pour le moins ingrate, j'ai modélisé l'univers de ma machine de la même manière que vous modélisez le monde réel, selon mon point de vue, un point de vue d'humain. De telle sorte que maintenant je vous perçois selon mon point de vue, selon mon modèle, les concepts de mon modèle reflètent l'univers de mon ordinateur. J'ai implémenté mon modèle sous la forme d'un langage de programmation et j'ai crée un petit programme que j'ai appelé compilateur, celui-ci réduit l'abstraction que j'ai crée en modélisant mes ressources informatiques. Il traduit mon langage en langage machine. Récemment j'ai modélisé l'univers de ma machine en objets. Je vois ma machine comme un monde d'objets et j'ai mis en place une série de modèles, de langages orienté objet. Mais attention ne croyez pas pour autant que vous pourrez prendre ma place maintenant qu'il existe des concepts plus perceptible, le modèle que j'ai mis en place n'est qu'une abstraction de l'univers du calculateur, ce n'est pas mon point de vue mais bien celui de ma machine, chaque objet est une structure atomique représentant des données et des traitements. Mais je sais que vous êtes généreux et trop occupé pour prendre ma place, aussi laissez moi vous dire ce dont je suis capable. Avec mes langages je modélise vos modèles. Lorsque je suis passé en orienté objet et que j'ai vu le monde de mon ordinateur de la même manière que vous vous voyez le monde réel c'est-à-dire en objets, j'ai pu définir la virtualisation. Mes objets désormais sont des représentations abstraites, des clones virtuels de vos objets, de vos concepts. Mes objets sont caractérisé par un état et un comportement, mes objets interagissent ils s'envoient des messages, ils communiquent. Les objets actifs qui envoient des messages, je les appelés acteurs. Les objets passifs qui ne font que recevoir des requêtes je les appelés serveurs. Ceux de mes objets qui reçoivent et qui émettent des messages je les appelés agents. Ces derniers sont comme des humains ils peuvent interagir avec les autres objets à tout moment, de leur propre initiative ou suite à une sollicitation externe. Ces objets je m'en sers pour simuler les êtres vivants et les communautés de vos modèles. Et depuis que vous mes très cher collègues avez observé le monde et avez déterminé qu'il existe une intelligence collective, de groupe, une intelligence émergeant des interactions d'êtres vivants, qui dans leurs individualité ne sont pas intelligent ; depuis cette découverte j'ai eu, en ce qui me concerne plus d'intérêt pour les agents. En simulant ces communautés d'êtres vivants avec mes agents, j'allais ajouter à mes méthodes de résolution classiques des méthodes intelligentes. J'allais évoluer vers un domaine qui me passionne depuis longtemps : rendre ma machine intelligente. Désormais la simulation informatique avait deux finalités : satisfaire les modélisateurs ou simuler l'intelligence. Dans les deux cas, il s'agissait de modéliser le monde réel, le vivant. Pour formaliser cette exercice, j'ai introduit une nouvelle catégorie de logiciel : les plateformes de modélisation et de simulation. Parce que j'ai constaté vos besoins en simulations sociales et

environnementales et parce que les agents représentent pour moi un outil remarquable dans le domaine de l'intelligence artificielle, j'ai décidé de mettre en place des systèmes multi-agents.

Afin de réaliser ce projet, j'ai voulu définir ce qu'était un système multi-agents afin d'en implémenter la définition. Mais, comme cela était prévisible, nous, les informaticiens n'avons pu nous fixer une définition unique. La désapprobation dans une communauté scientifique fait le charme de celle-ci. Pour résoudre ce nouveau problème j'ai usé une fois de plus de ma capacité d'abstraction. Des définitions qui ont été émise je n'ai retenu que ceci : un système multi-agents est un système complexe. Or un système complexe est un ensemble de points de vue, un ensemble de modèles en interactions. J'ai compris qu'un système multi-agents, peu importe la définition que l'on en fait n'est en fait qu'un système complexe spécialisé. Et comme je savais que jamais les divergences en termes de définitions ne cesseraient, j'ai préféré implémenter la définition d'un système complexe en laissant la liberté à tout un chacun de le décrire, de le définir. J'ai ainsi défini une structure générique de modélisation, qui allait permettre aux différentes définitions de cohabiter. J'ai défini les plateformes génériques de modélisation et de simulation. »

D.Les systèmes multi-agents

1. Qu'est ce qu'un agent ?

Le professeur Ferber [1997] nous donne une définition d'un agent :

On appelle agent une entité physique ou virtuelle

- qui est capable d'agir dans un environnement
- qui peut communiquer directement avec d'autres agents.
- qui est mue par un ensemble de tendance (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),
- qui possède des ressources propres
- qui est capable de percevoir (mais de manière limitée) son environnement,
- qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
- qui possède des compétences et offres des services,
- qui peut éventuellement se reproduire, dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et

en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

2. Les systèmes multi-agents

Le professeur Ferbert nous donnes une excellente définition des systèmes multi-agents, un système multi-agents est constitué selon lui :

- Un environnement E, dans notre cas, c'est l'espace où peuvent se déplacer les animaux.
- Un ensemble d'objets O. Ces objets sont situés, c'est-à-dire que pour tout objet, il est possible, à un moment donné, d'associer une position dans E.
- Un ensemble A d'agents qui sont des objets particuliers (AÍO), lesquels représentent les entités actives du système.
- Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux.
- Un ensemble d'opérations Op permettant aux agents de A de percevoir, produire, consommer, transformer, et manipuler des objets de O. Cela correspond à la capacité des agents de percevoir leur environnement, de manger, etc. Nous reviendrons sur ce point particulièrement important.
- Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers

3. Pourquoi parler des systèmes multi-agents ?

Un système multi-agents est un système complexe. Sa modélisation nécessite une multiplication des points de vue comme cela est illustré par la définition précédente. Pour simuler un système multi-agents il est nécessaire de disposer d'une plateforme multi formalismes, qui dispose d'outils permettant de spécifier des interactions entres les modèles.

E.Les plateformes de modélisation et simulation existantes

Au niveau international, la plateforme multi-agents la plus connue, Swarm, a été réalisée dans le cadre du Santa Fe Institute, initialement par Chris Langton, puis maintenant par un ensemble de développeurs (Daniels 1999). Se présentant comme un ensemble de bibliothèques de classes écrites en Objective C (et accessibles maintenant depuis Java),

Swann a été très utilisé dans le monde anglo-saxon, surtout grâce à son efficacité en termes de temps de réponse et donc à sa capacité à gérer des simulations de taille importante.

Malheureusement, elle est très peu puissante sur le plan de l'expressivité et très difficile à mettre en œuvre par un non informaticien. De ce fait, elle reste utilisée essentiellement pour la simulation de phénomènes physicochimiques ou biologiques élémentaires, où la vitesse d'exécution est un élément fondamental tout en ne nécessitant pas la définition de modèles trop complexes. Enfin elle souffre de nombreux défauts sur le plan conceptuel et architectural, ce qui la rend peu attractive pour modéliser des phénomènes faisant intervenir à la fois des entités biologiques et des individus sociaux. Dans la communauté internationale des chercheurs travaillant sur la modélisation par systèmes multi-agents (forum SimSoc par exemple), les manques et défauts de Swarm sont reconnus.

Au niveau francophone les principales plateformes sont :

- la plateforme CORMAS, développé par le CIRAD de Montpellier. CORMAS est une plate-forme générique pour la simulation multi-agents. Cette plate-forme, qui est disponible en ligne (http://cormas.cirad.fr), a été utilisée dans de très nombreuses modélisations. Ses facilités d'interaction avec l'utilisateur, sa simplicité d'apprentissage, ses fonctions de modélisations intégrées, lui ont permis d'être plébiscitée par un très grand nombre d'utilisateurs.

- MADKIT est une plate-forme générique développée au LIRMM, elle aussi est disponible en ligne (http://www.madkit.org). Ses caractéristiques essentielles sont la généricité, la modularité, l'extensibilité, la prise en compte de nombreux types d'environnements et de mode de modélisation. De plus, il est possible de faire tourner cette plate-forme en réseau pour des raisons d'efficacité ou d'utilisation à plusieurs.

D'autres plates-formes existent, mais n'offrent pas le même niveau de généricité que les deux précédentes. Citons notamment :

- MobyDick, plate-forme développée à l'INRA, est utilisée pour la simulation individu-centrée et tout particulièrement pour modéliser l'évolution biologique (développement, utilisation des ressources, reproduction) des espèces naturelles et plus particulièrement des poissons.

- MANTA développée par le LIP6 dans les années 90, est une plate-forme dévolue à la modélisation de comportements de fourmis à partir d'une approche éthologique.

Il existe aussi un grand nombre de plateformes non génériques destinées à la réalisation d'un seul modèle. C'est encore le cas de nombreuses simulations qui sont réalisées lors d'un projet et utilisées uniquement dans le cadre de ce projet et qui n'offrent de ce fait pas des outils de constructions de modèles suffisamment génériques.

F.La plateforme MIMOSA

L'objectif de Mimosa est de spécifier les outils génériques permettant aux modélisateurs de décrire leurs modèles ainsi que les simulations. La plupart des plateformes de modélisation et de simulation définissent les formalismes dans lesquels le modèle est exprimable (STELLA (Tilideske, 1998), DEVS (Zeigler et al, 1999),CORMAS (cormas,2003), STARLOGO(starlogo, 2002), etc.) ou laissent partiellement la liberté de définir ces formalismes, mais en donnant l'accès direct au langage de programmation sous-jacent. Le principe est le suivant :

Un modèle est un discours sur l'expérience. Ce discours articule des concepts, que ces concepts réfèrent à des objets individuels (on parlera de concepts individuels) ou à des catégories d'objets individuels (on parlera alors de concepts catégoriels). Nous appellerons modèle ou point de vue un ensemble de concepts.

Ces discours constituent autant de formalismes au sens large, c'est-à-dire pas mathématique tant que ces formalismes ont un caractère formel au sens qu'ils ont une forme, donc une syntaxe.

En conséquence, un outil générique de modélisation doit permettre de définir à la fois les moyens du discours (les formalismes) et le discours lui-même (les concepts individuels et catégoriels). La modélisation d'un système complexe, par exemple un écosystème, nécessite la multiplication des points de vue (écologique, agronomique, sociologique, économique…) qui sont autant de discours ayant chacun leur spécificité de par le formalisme utilisé et les concepts énoncés dans ce formalisme, mais qu'il s'agit d'articuler. En passant, nous insistons sur l'articulation plutôt que sur l'intégration qui supposerait ou imposerait un discours universel et ultime dans lequel tout pourrait s'exprimer.

L'objectif de MIMOSA se décline en la mise en place d'outils permettant de spécifier les formalismes utilisés et les modèles exprimés dans ces formalismes en faisant l'hypothèse que tout formalisme (donc tout modèle) peut se décrire en termes de composants et de composés. Nous entendons donc à ce stade la généricité comme la possibilité d'être multi formalisme, multi modèle, (on dit aussi multi point de vue) et multi niveau. Cette généricité se veut issue d'une réflexion sur la modélisation, et plus généralement de représentation des connaissances dans la mesure où l'intelligence artificielle et l'informatique proposent également des formalismes de représentation autres que les formalismes strictement mathématiques.

Nous proposons donc une boîte à outil permettant de décrire une vaste gamme de formalismes, puis les modèles exprimés dans ces formalismes et de les articuler entre eux

en vue de modéliser des systèmes complexes. Après un travail comparatif sur les différents formalismes utilisés dans la littérature, nous faisons l'hypothèse que cette boîte à outil peut être constituée de notions très simples permettant de décrire les structures et les processus, à savoir :

- Les composants, les composés et les relations pour les descriptions structurelles
- Les mesures, les évènements, les états et les fonctions de transition pour les descriptions dynamiques

Chapitre II.
Les outils

A.La programmation orientée objet (POO)

Les informaticiens ont récemment modélisé les ressources informatiques en termes d'objets. Où l'objet est une fusion entre un état et un comportement. Ils ont pu simuler le monde réel en se servant des objets informatiques comme des représentations abstraites des entités du monde réel. « Ces objets encapsulent une partie de la connaissance du monde réel ». Les langages de programmation orientées objet constituent des implémentations de cette modélisation.

B.Le langage Java

MIMOSA et S-Edit ont été développé dans le langage Java du fait de l'indépendance vis-à-vis des plateformes de ce langage. Le langage java est le dernier-né des langages objet. Il s'inspire des langages C et C++ dont il conserve la concision tout en supprimant les aspects les plus confus et les plus générateurs d'erreurs. Il s'inspire aussi du SmallTalk en ce qui concerne la machine virtuelle. Initialement conçu pour la programmation des assistants électroniques, le langage Java s'est vu propulser sur le devant de la scène pour ses potentialités de création de contenu dynamique pour l'Internet.

C.Le formalisme UML

UML, acronyme de Unified Modeling Language est un langage graphique de modélisation objet. Ce langage est composé d'une large gamme de formalismes visuels permettant de décrire un monde d'objet. Parce que les informaticiens ont décrit leur univers en terme d'objets, UML est devenu pour cette communauté un outil de base pour la description, la spécification, la construction, la visualisation des systèmes logiciels. UML est un langage de modélisation et non une méthode. Au travers de ce mémoire nous utiliserons un certain nombre de diagrammes, à savoir le diagramme de classe, le diagramme des cas

d'utilisations, le diagramme de séquence, les collaborations paramétrables et le diagramme d'état transition.

Nous avons préféré présenter ces diagrammes dans un chapitre séparé car nous en utiliserons les concepts pour modéliser dans ce document tout ce qui est objet ; objet dans le sens général et non dans le sens informatique.

1. Le diagramme de classes

Les diagrammes de classes expriment de manière générale la structure statique d'un système, en termes de classes et de relations entre ces classes. Outre les classes, ils présentent un ensemble d'interfaces et paquetages, ainsi que leurs relations.

a) Les classes

La classe décrit le domaine de définition d'un ensemble d'objets. Chaque objet appartient à une classe. Les généralités sont contenues dans la classe et les particularités sont contenues dans les objets. Les objets informatiques sont construits à partir de la classe, par un processus appelé instanciation. Les classes sont représentées par des rectangles compartimentés. Le premier compartiment contient le nom de la classe qui est unique dans le paquetage qui contient cette classe. Deux autres compartiments sont généralement ajoutés ; ils contiennent respectivement les attributs et les opérations de classe. Les compartiments d'une classe peuvent supprimés lorsque leur contenu n'est pas pertinent dans le contexte d'un diagramme.

Certains attributs et opérations peuvent être visibles globalement, dans toute la portée lexicale de la classe (par défaut la portée est celle de l'instance). Ces éléments également appelés attributs et opérations de classe, sont représentés comme un objet avec leur nom souligné.

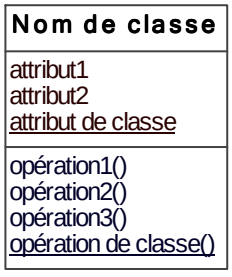
Représentation simplifiée	Représentation complète
	

Figure 1. *Représentation graphique des classes*

b) Les associations

Les associations représentent des relations structurelles entre classes d'objets. Une association symbolise une information dont la durée de vie n'est pas négligeable par rapport à la dynamique générale des objets instances des classes associées. Une association compte au moins deux extrémités d'association, reliées à des classificateurs. Les associations se représentent en traçant une ligne entre les classes associées.

Les associations peuvent être nommées ; le nom de l'association figure alors au milieu de la ligne qui symbolise l'association, plus précisément au dessus, au-dessous ou sur la ligne en question.

L'extrémité d'une association possède un nom. Ce nom, aussi appelé rôle, décrit comment une classe source voit une classe destination au travers de l'association. Chaque association binaire possède deux rôles, un à chaque extrémité. Un rôle se présente sous forme nominale. Le nom d'un rôle se distingue du nom d'une association, car il est placé près d'une extrémité de l'association.

Chaque extrémité d'une association peut porter une indication de multiplicité qui montre combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe. La multiplicité est une information portée par l'extrémité d'association, sous la forme d'une expression entière.

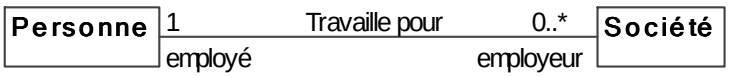


Figure 2. *Représentation graphique d'une association*

c) Les agrégations

Une agrégation représente une association non symétrique dans laquelle une des extrémités joue un rôle prédominant par rapport à l'autre extrémité. Quelle que soit l'arité, l'agrégation ne peut concerner qu'un seul rôle d'une association. L'agrégation se représente en ajoutant un petit losange du coté de l'agrégat.

Figure 3. *Représentation graphique d'une agrégation*

d) Les compositions

La composition est un cas particulier d'agrégation avec un couplage plus important. La classe ayant le rôle prédominant dans une composition est appelée classe composite ou classe conteneur. La composition implique, en plus des propriétés

d'agrégation, une coïncidence des durée de vie des composants et du composite : la destruction du composite implique automatiquement la destruction de tous ses composants. La création, la modification et la destruction des divers composants sont de la responsabilité du composite. Dans une composition, un composant ne peut être partageable, d'où la contrainte sur la multiplicité du coté de l'agrégat qui ne prendre que les valeurs de 0 ou de 1. La composition se représente par un losange de couleur noire, placé du côté de la classe composite.

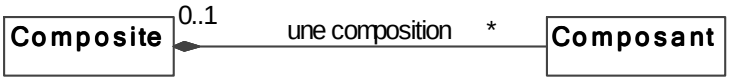


Figure 4. *Représentation graphique de la composition*

e) La généralisation

La relation de généralisation est une relation de classification entre un élément plus général et un élément plus spécifique. Par exemple, un animal est un concept plus général qu'un chat ou un chien. La relation de généralisation se représente au moyen d'une flèche orientée de la classe la plus spécialisée vers la classe la plus générale. La tête de la flèche possède un petit triangle vide.

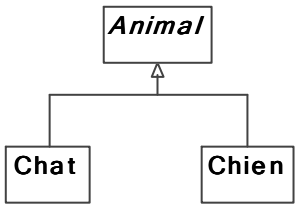


Figure 5. *Représentation graphique de la relation de généralisation entre classes*

2. Le diagramme de cas d'utilisation

Les cas d'utilisations décrivent, sous la forme d'actions et de réactions le comportement d'un système du point de vue d'un utilisateur. Ils permettent de définir les limites système et les relations entre le système et l'environnement.

a) Les acteurs

Les acteurs se représentent sous la forme de petits personnages. Un acteur représente un rôle joué par une personne ou une chose qui interagit avec un système.

b) Les cas d'utilisation

Un cas d'utilisation est un classificateur qui modélise une fonctionnalité d'un système ou d'une classe. L'instanciation d'un cas d'utilisation se traduit par l'échange de messages entre le système et ses acteurs.

Les cas d'utilisation peuvent être contenus dans un rectangle qui représente les limites du système ; Les acteurs sont alors forcément à l'extérieur du rectangle puisqu'ils ne font pas partie du système.

La relation entre un acteur et un cas d'utilisation est une association représentée par une ligne.

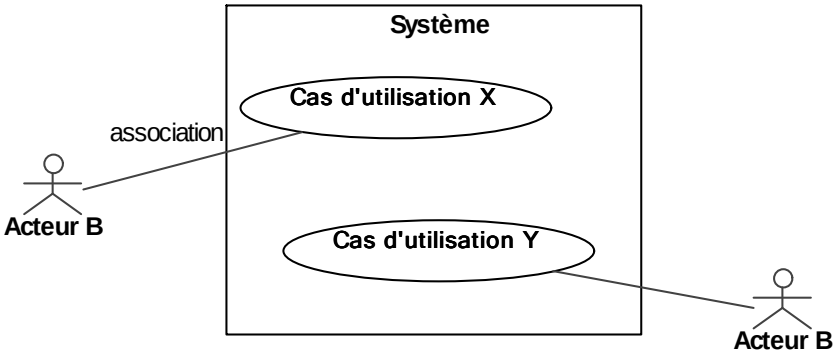


Figure 6. *Représentation graphique des cas d'utilisation*

c) La relation d'inclusion

Dans une relation d'inclusion entre cas d'utilisation, une instance du cas d'utilisation source comprend également le comportement décrit par le cas d'utilisation destination. L'inclusion a un caractère obligatoire, la source spécifiant à quel endroit le cas d'utilisation cible doit être inclus. Cette relation permet ainsi de décomposer des comportements et de définir des comportements partageables entre plusieurs cas d'utilisation. Elle est représentée par une flèche en trait pointillés pointant vers le

cas d'utilisation de destination et par le mot clé « «include » placé à proximité de la flèche.

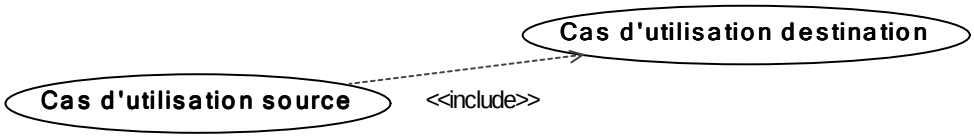


Figure 7. *Représentation graphique de la relation d'inclusion*

3. Le diagramme de séquence

Les diagrammes de séquence montrent des interactions entre objets. La représentation se concentre sur la séquence des interactions selon un point de vue temporel. Les diagrammes de séquence sont aptes à modéliser les aspects dynamiques des systèmes temps réel et des scénarios complexes mettant en œuvre peu

a) Les interactions

Une interaction modélise un comportement dynamique entre objets. Elle se traduit par l'envoi de message entre objets. Un diagramme de séquence représente une interaction entre objets, en insistant sur la chronologie des envois de message.

Un objet est matérialisé par un rectangle et une barre verticale, appelée ligne de vie des objets.

Les objets communiquent en échangeant des messages représentés au moyen de flèches horizontales, orientées de l'émetteur du message vers le destinataire.

La dimension verticale représente l'écoulement du temps (de haut en bas, par défaut). L'ordre d'envoi des messages est ainsi donné par la position de ces messages sur les lignes de vie des objets.

b) Les activations et envois de messages

Les diagrammes de séquence permettent également de représenter les périodes d'activité des objets. Une période d'activité correspond au temps pendant lequel un objet effectue une action, soit directement, soit par l'intermédiaire d'un autre objet qui lui sert de sous-traitant. Les périodes d'activité se représentent par des bandes rectangulaires placées sur les lignes de vie. Le début et la fin d'une bande correspondent respectivement au début et à la fin d'une période d'activité.

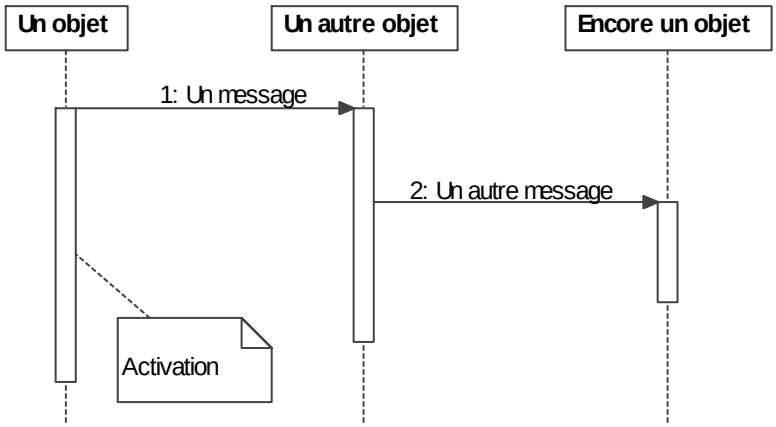


Figure 8. *Un digramme de séquence*

D.Le format XML

XML (*Extensible Markup Language* ou *langage de balisage extensible*) est un standard du World Wide Web Consortium qui sert de base pour créer des langages de balisage spécialisés : c'est un « méta-langage ». En ce sens, XML permet de définir un vocabulaire et une grammaire associée sur base de règles formalisées. Il est suffisamment général pour que les langages basés sur XML, appelés aussi dialectes XML, puissent être utilisés pour décrire toutes sortes de données et de textes. Il s'agit donc partiellement d'un format de données. L'extensibilité de XML est principalement assurée par la notion fondamentale d'espaces de noms (NameSpace).

Les dialectes XML sont décrits de façon formelle : une structure de données simple est définie avec une DTD (*Document Type Definition*), une structure de données détaillée est définie avec un XML Schema ou tout autre DSDL (*Document Schema Definition Languages*, c'est-à-dire langage de description de schéma).

1. Le fichier XML

Un fichier XML est un fichier texte. Le codage des caractères est défini dans la première déclaration du document. Par défaut il s'agit de UTF-8, une transcription binaire particulière de Unicode, un format qui diffère peu de l'ASCII.

Le fichier XML est structuré en éléments à l'aide de balises qui marquent le début et la fin de chaque élément. Les éléments peuvent contenir du texte et éventuellement d'autres éléments. L'ensemble des données du document XML est contenu dans un élément unique appelé racine, élément qui contient tous les autres éléments.

Outre les éléments et le contenu des éléments, on trouve aussi dans un document XML:

- des commentaires (qui ne font pas partie des données au sens strict) ;
- des instructions de traitement (directives données au processeur XML) ;
- des « appels » de caractère (pour coder des caractères qui n'existent pas dans le codage choisi pour le document tout entier) ;
- des « appels » d'entités (permettent l'appel d'une entité nommée qui est une sorte de « macro » de texte).

2. La DTD

Le **Document Type Definition** (DTD), ou **Définition de Type de Document**, est un document permettant de décrire un modèle de document SGML ou XML. Une DTD ne décrit cependant que la structure du document (hiérarchie des champs, paramètres, type des données...) et non, par exemple, les valeurs autorisées des champs ou paramètres, ce en quoi elle se distingue de Schéma XML. De plus, la norme DTD fait appel à une syntaxe spécifique distincte de XML. Une DTD n'est donc pas un document XML.

3. Les API Java pour XML

Une API est une interface de programmation applicative (application programming interface). Il s'agit d'une interface normalisée permettant à un logiciel de faire appel aux fonctions d'un autre programme déjà disponible sur une machine. Concrètement, les API que nous allons voir par la suite vont permettre au programmeur Java de gérer facilement et efficacement l'accès aux fichiers XML en faisant appel à des méthodes prédéfinies. Java possède, depuis le JDK1 1.4, deux API permettant la gestion de fichiers XML regroupées dans le package JAXP (Java API for XML Parsing). On peut trouver cette API séparément sur le site de Sun Microsystem – http://java.sun.com .

4. L'API SAX

SAX (Simple API for XML) est basé sur un modèle événementiel, ce qui signifie que l'analyseur appelle automatiquement une méthode lorsqu'un événement est détecté dans le fichier XML.
Evénement signifie, par exemple, détection d'une balise ouvrante, de la fin du document etc…
Voici les 5 évènements détectés par SAX ainsi que les méthodes qui sont appelées :

- Détection d'une balise de début ***startElement()***
- Détection d'une balise de fin ***endElement()***
- Détection de données entre deux balises ***characters()***
- Début du traitement du document XML ***startDocument()***
- Fin du traitement du document XML ***endDocument()***

5. L' API DOM

La programmation en utilisant DOM (Document Object Model) est totalement différente de la méthode SAX. En effet, le document n'est plus lu de façon linéaire mais il est

parcouru plusieurs fois par l'analyseur et stocké entièrement en mémoire sous forme d'un arbre que l'on appelle arbre DOM. Contrairement à SAX, cette API présente l'intérêt de pouvoir tenir compte de la hiérarchie des éléments, et d'ajouter, supprimer ou modifier des éléments de cet arbre.

6. L'avantage du SAX

L'avantage du SAX sur le DOM réside principalement sur la vitesse de lecture et l'économie de mémoire car le SAX ne construit pas une représentation en mémoire de la structure en arborescence des fichiers XML. Cette représentation en mémoire peut dans certains cas s'avérer extrêmement critique, dépendamment de la complexité des fichiers traités et des besoins.

La plateforme MIMOSA et le système S-Edit utilisent le format XML pour définir un vocabulaire de description des formalismes. Ces vocabulaires définissent des bases indépendantes pour ces systèmes à partir desquelles les modélisateurs vont pouvoir décrire leurs modèles. Dans l'annexe B de ce document vous pouvez retrouvez la DTD du système S-Edit.

Chapitre III. Dossier technique

A.La notion d'éditeur de modèle

D'après l'énoncé du concept de modèle retenu par MIMOSA, un éditeur de modèle est une entité logicielle disposant d'une interface permettant d'instancier des composants, des relations et des composés. L'interface est une interface entre MIMOSA et un humain. L'humain ne peut interagir qu'avec ce qu'il perçoit. Parmi les perceptions de l'humain la vue occupe une place importante. Or les concepts des modèles que l'humain doit articuler ne sont pas forcément exprimés dans des formalismes visualisables. Ce qui implique que pour que l'humain puisse interagir, puisse utiliser les concepts d'un modèle, il est nécessaire pour ces concepts de disposer d'une forme perceptible. En l'occurrence une forme graphique. D'autant plus qu'un modèle est un graphe, or les graphes (voir *théorie des graphes*) disposent d'un formalisme de représentation graphique. Par conséquent un éditeur de modèle doit reposer sur un système qui associe aux formalismes du modèle un formalisme perceptible, un formalisme graphique si l'éditeur est graphique.

La généricité d'une plateforme de modélisation repose sur la possibilité de définir librement les formalismes dans lesquels les modèles sont exprimés. Ainsi une plateforme générique doit disposer d'un système de définition des formalismes. Cette description se fait par programmation, le système de définition étant un ensemble de règles et de directives. Lorsqu'on définit un nouveau formalisme, on peut avoir besoin de définir de nouvelles formes graphiques pour représenter les concepts de ce formalisme. On peut avoir besoin de définir un nouveau formalisme graphique. Cela suppose qu'un système de définition de formalisme graphique ait été établi. Cela suppose l'existence d'un mécanisme d'extension des classes de définitions des formalismes graphiques. Si les composants, les composés, les relations constituent une boite à outils permettant de décrire tout modèle, si un modèle est un graphe et que les graphes disposent d'un formalisme de représentation alors les éléments de ce formalisme constituent une boite à outils graphique permettant de décliner toute visualisation de modèle. Et cela constitue la condition indispensable pour l'édition de modèles dans une plateforme générique.

Aussi avons-nous retenu la définition suivante: un éditeur graphique de modèle est un outil logiciel permettant d'éditer des modèles par l'intermédiaire de l'édition d'un modèle graphique dont le formalisme est associé à celui du modèle. Le modèle graphique utilise comme formalisme de base le formalisme de représentation des graphes. Cet éditeur doit reposer sur un système d'association et de définition des formalismes graphiques.

Dans l'état actuel, l'éditeur de modèle, dans la définition que nous venons d'en faire, n'est pas encore implémenté dans la plateforme MIMOSA. Le système d'édition actuelle fait abstraction de la représentation graphique des concepts ou du moins il se sert de leur représentation textuelle. La figure suivante illustre l'édition d'un modèle de population. Une édition certes fonctionnelle mais peu intuitive et souvent inadaptée dés lors qu'il s'agit de représenter l'état des composants.

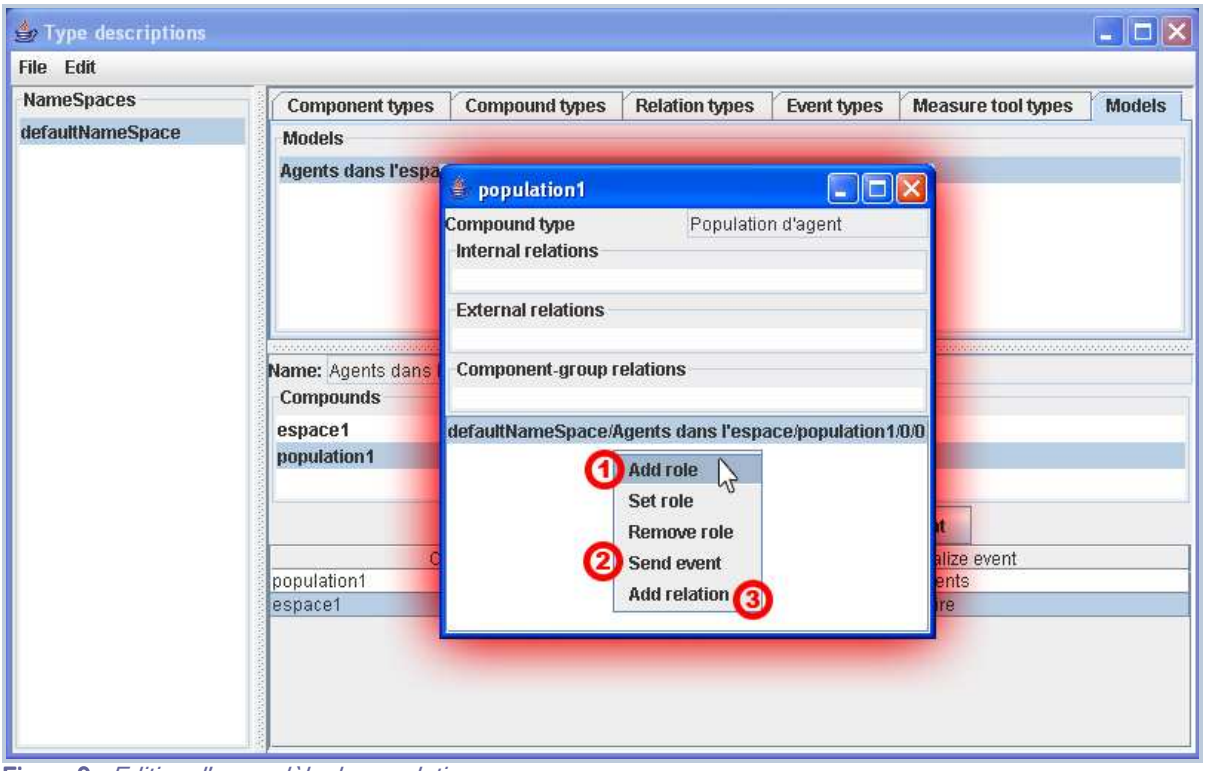


Figure 9. *Edition d'un modèle de population*

Pour palier à ce manque, les programmeurs de formalisme sont obligés d'élaborer des systèmes complets de visualisation d'édition propres à un formalisme donné (figure ci-contre). Ils ne disposent pas d'une boite à outils leurs permettant de décrire l'édition. Boite à outils, car toute démarche d'abstraction aboutit à la création d'outils. Réaliser un éditeur de modèle pour une plateforme générique tel que MIMOSA revient à concevoir une abstraction des démarches classiques d'édition de modèles.

Le système S-Edit implémente la définition que nous avons établie pour un éditeur générique de modèle. Le projet MIMOSA s'inscrivant dans une démarche collaborative et la plateforme MIMOSA devant être une

synthèse du meilleur de l'existant technique, le couplage entre la plateforme et S-Edit est une idée relativement intéressante.

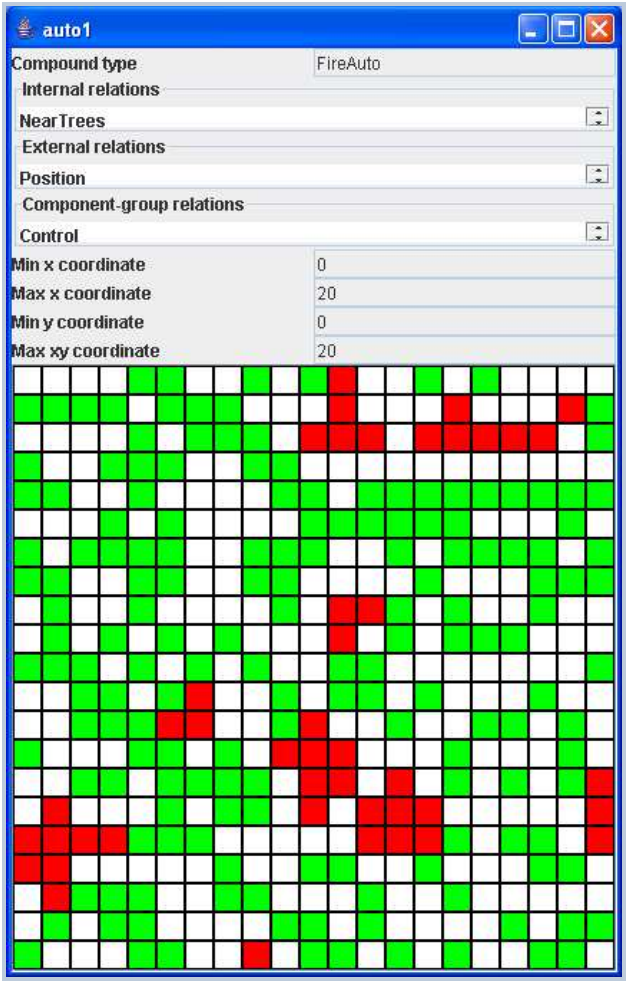


Figure 10. *Edition d'un automate cellulaire*

B.Ce qu'il s'agit de faire

Un couplage est une mise en relation de deux systèmes. L'intérêt évident du couplage est que le système résultant hérite des fonctionnalités combinées de deux systèmes. Tout système dispose de fonctionnalités de base, les fonctionnalités pour lesquelles il est crée et pour lesquelles il est utilisé.

La lecture des objectifs du projet MIMOSA permet dans une phase conceptuelle de dégager un certain nombre de fonctionnalités. Les fonctionnalités essentielles de la plateforme MIMOSA concourent à permettre une modélisation générique. Parmi elles on compte :

- La définition du formalisme des modèles : cette fonctionnalité est à la base de la généricité de la plateforme puisqu'elle permet une libre définition du formalisme des modèles. L'interfaçage entre cette fonctionnalité et l'utilisateur est effectué à travers le langage de programmation. Cette fonctionnalité n'est donc utilisée pour le moment que par les programmeurs de formalismes. Ceux-ci accèdent à des notions asémantiques, des outils dont ils se servent pour décliner des formalismes;

- La description interactive des catégories d'objets du discours : cette fonctionnalité permet au modélisateur de décrire son modèle pendant l'exécution du logiciel générique MIMOSA. Il faut distinguer le logiciel générique de la plateforme : le logiciel générique est inclus dans la plateforme au même titre que le langage de programmation Java;
- La construction et la simulation de modèles concrets : cette fonctionnalité permet au modélisateur d'instancier les catégories d'objets du discours afin de constituer un modèle. Cette fonctionnalité utilise, entre autres, une fonctionnalité d'édition générique des modèles ;
- L'édition générique des modèles : cette fonctionnalité permet d'éditer les modèles indépendamment du formalisme dans lesquelles ils sont exprimés. Cette fonctionnalité permet au programmeur de formalisme, futur modélisateur de ne pas se soucier de l'interface d'édition des concepts définis dans son formalisme. Cette fonctionnalité n'est pas implémentée, notre tache consiste à utiliser S-Edit comme implémentation de cette fonctionnalité.

Ces fonctionnalités sont schématisées dans le diagramme des cas d'utilisation suivant. La coloration blanche d'un cas d'utilisation dénote de l'état son implémentation.

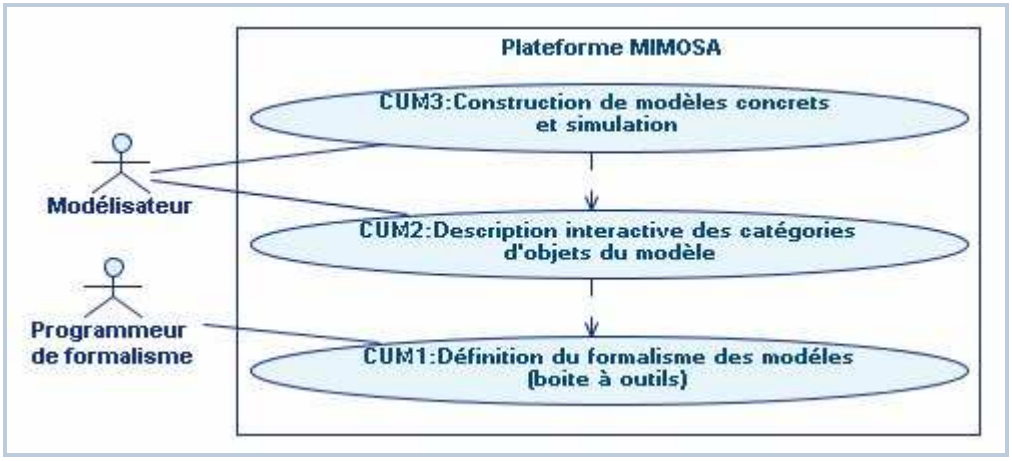


Figure 11. *Fonctionnalités de la plateforme MIMOSA*

Les fonctionnalités essentielles du système S-Edit permettent une édition générique des modèles. De façon identiques à MIMOSA elles sont constitué de :

- La définition par programmation du formalisme des modèles : elle inclue ici la définition du formalisme du modèle graphique. Le programmeur décline une notion graphique de base afin de constituer une forme graphique, un formalisme graphique. On retrouve une boite à outils comme dans MIMOSA si ce n'est qu'elle est enrichie d'un outil supplémentaire : une notion graphique asémantique;

- La description statique des catégories d'objets du discours : contrairement à MIMOSA cette description se fait au travers de fichiers XML préchargés. Cette description comprend l'association à une catégorie d'objet graphique qu'il convient ensuite de paramétrer, de décrire ;
- L'édition de modèle telle que nous l'avons définie au début de ce chapitre. Cette fonctionnalité est composée des sous fonctionnalités suivantes :
  - La génération d'une barre d'outils permettant d'instancier les catégories d'objets du modèle ;
  - L'édition de diagramme : les symboles utilisé par les diagrammes représentent les instances des catégories d'objets du modèle ;
  - L'inspection de propriétés:cette fonctionnalité permet de modifier l'état des concepts à travers un inspecteur de propriétés ;
  - L'exécution des actions définies sur les objets du discours, cette fonctionnalité permet de simuler les modèles.

Le diagramme suivant illustre ces fonctionnalités.

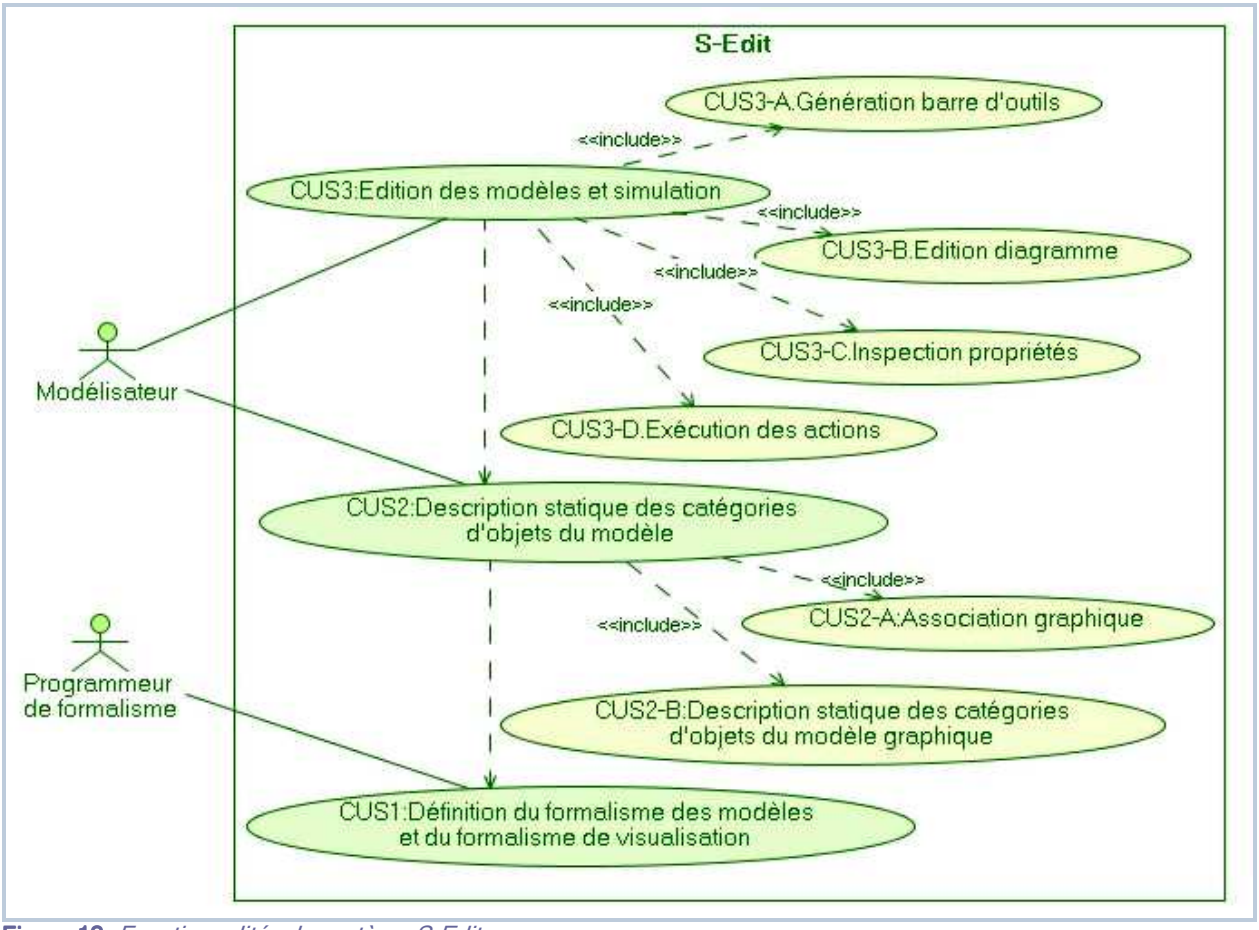


Figure 12. *Fonctionnalités du système S-Edit*

La modélisation informatique c'est l'implémentation d'un modèle sur ordinateur pour en exploiter le caractère prédictif lors de simulations mettant à profit la puissance de calcul de l'ordinateur. Un formalisme est un ensemble de définition des concepts articulés par un modèle. Dans la modélisation informatique, le formalisme dans son implémentation dans l'applicatif Java, est un ensemble de classes. Le modèle est une instanciation de ces classes. Le système de conception de formalisme permet de concevoir ces classes.

La généricité est le résultat d'un processus d'abstraction. Ce résultat est servi par deux mécanismes en programmation orienté objet : la généralisation et l'interfaçage. En effet, un modèle concret est implémenté sur ordinateur selon un paradigme objet. Un autre modèle est ensuite implémenté. Le premier et le deuxième modèle définissent des notions différentes à travers leurs formalismes. Après observation on se rend compte que les définitions des notions de part et d'autres comportent des similitudes. Ces notions qu'elles soient structurelles ou dynamiques sont implémentées par des objets. La programmation orienté objet nous propose alors d'implémenter les similitudes dans des classes mères, et les divergences dans des classes filles. Il s'agit de la généralisation. Les classes mères implémentent alors des notions que l'on qualifie de notions de base ; les classes filles implémentent la spécificité du modèle. Spécificité du modèle par rapport à tout les autres modèles. Les classes mères ne contenant aucune spécificité, elles implémentent des notions de base **asémantiques**. Les notions des deux modèles deviennent des déclinaisons de notions de base asémantiques. Ces notions, leur implémentations ne définissent pas le formalisme de tout les modèles mais permettent de définir tous les formalismes : ces classes sont des outils. Aussi elles constituent la boite à outils asémantique de tout système générique. Cette boite à outils confèrent à MIMOSA et à S-Edit la fonctionnalité commune de libre définition du formalisme des modèles (**CUx1**). Les déclinaisons de cette boite à outils constituent les modèles.

Ce qu'il s'agit de faire c'est : faire migrer la fonctionnalité d'édition de modèle du système S-Edit vers la plateforme MIMOSA. De sorte que l'éditeur de modèle du système S-Edit puisse construire des modèles issus de la déclinaison de la boite à outils MIMOSA. En d'autre terme, si l'on modélisait la plateforme MIMOSA selon le pattern **modèle – vue – contrôleur** le système S-Edit serait le contrôleur et les déclinaisons de la boite à outils seraient le modèle.

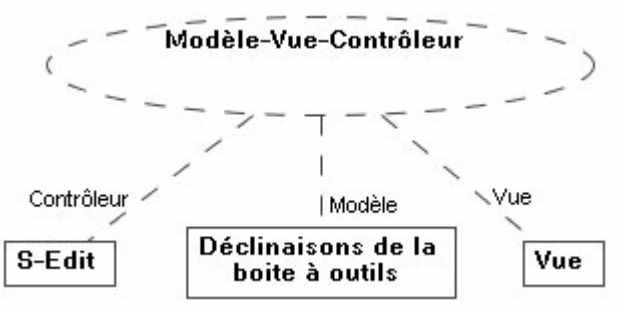


Figure 13. *Modélisation de la plateforme MIMOSA selon le pattern MVC*

Automatiquement le problème suivant se pose : MIMOSA et S-Edit utilisent-ils la même boite à outils ? Leurs boite à outils sont-elles issus de la même démarche d'abstraction des modèles informatiques ? Leurs boite à outils utilisent-elles les mêmes définitions de modèles ? De la définition des modèles découle la structure statique et dynamique du modèle. S'ils utilisent la même définition, utilisent-ils la même implémentation de la boite à outils : les mêmes classes, les mêmes interfaces ? Certainement non. Alors quelles solutions ?

C.Généricité, boites à outils, formalismes

1. Concepts de modèles, abstractions et définitions

a) Notions structurelles et notions dynamiques

Le monde évolue, il est soumis au temps. Le modèle étant une simplification, une idée, une abstraction du monde, il doit rendre compte de l'évolutivité du monde. Les notions structurelles permettent de décrire le modèle. Les notions dynamiques permettent de décrire l'évolution du modèle.

b) La boite à outils MIMOSA

La boite à outils des systèmes génériques de modélisation est issu d'une démarche d'abstraction des modèles. Une définition est une abstraction. La boite à outils de la plateforme MIMOSA implémente d'une part :

- La définition de Maturana du modèle. Maturana définit le modèle comme étant : un discours sur l'expérience, que cette expérience soit produite par les canons de la science ou de la vie quotidienne. Ce discours articule des concepts, que ces concepts réfèrent à des objets individuels (on parle de concepts individuels) ou à des catégories d'objets individuels (on parle alors de concepts catégoriels). Un modèle ou point de vue étant un ensemble de concepts. Ce, ou plutôt ces discours se donnent les moyens pour le dire : compartiments, équations, champs,

objets, agent et que ces moyens sont très variés. Ils constituent autant de formalismes, qu'il faut entendre au sens large, c'est-à-dire pas seulement mathématiques tant que ces formalismes ont un caractère formel au sens qu'ils ont une forme, donc une syntaxe.

- D'autre part l'abstraction de la modélisation des systèmes complexes tels que les éco-sociosystèmes. En effet la modélisation de ces systèmes nécessite une multiplication des points de vue (écologique, agronomique, sociologique, économique) donc une multiplication des modèles. La modélisation informatique des systèmes complexes doit permettre d'observer les interactions, ce à des fins scientifiques ou politiques. Or une interaction est en fait une relation entre deux points de vue, entre deux modèles. La boite à outils de MIMOSA a été conçu pour permettre de modéliser des systèmes complexes et en particulier les interactions.

Aussi, résultat de cette observation de la démarche de modélisation, la boite à outils de la plateforme MIMOSA est constituée de plusieurs notions, des notions structurelles pour décrire les choses dont on veut parler et des notions dynamiques pour décrire les processus.

- **Les notions structurelles**

  MIMOSA propose les notions structurelles suivantes :

  - **La notion de composant** pour décrire une entité non décomposable ;
  - **La notion de composé** pour décrire un agrégat de composants. Un composé décrit comment il nomme ses composants ;
  - **La notion de relation** qui se décline en trois types de relations :
    - **Les relations internes ou intra-composés** qui décrivent des relations entre les composants d'un composé, donc qui sont définies entre les éléments d'un ensemble ;
    - **Les relations inter-composés** qui décrivent les relations entre deux composés, donc d'un ensemble dans un autre, elles servent à modéliser les articulations entre points de vue ;
    - **Les relations entre un composant et un composé** : il en existe une par défaut qui est la relation d'appartenance.

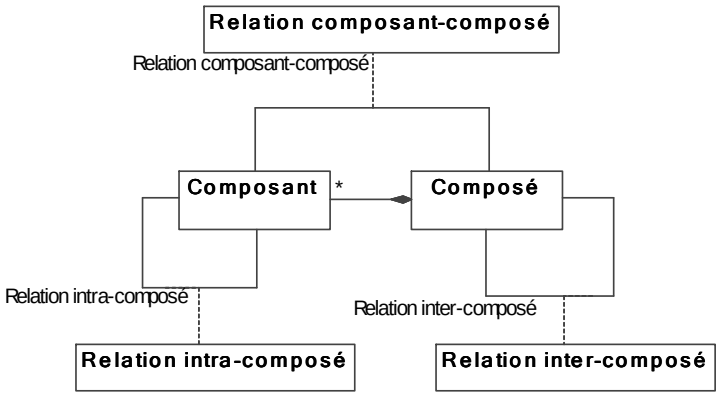


Figure 14. *Les notions structurelles de la boite à outil MIMOSA*

- **Les notions dynamiques**

  MIMOSA propose les notions dynamiques suivantes :

  - **Les états** : ils sont associés indifféremment aux composants et aux composés et permet de décrire l'état dynamique (Ce qui évolue au fil du temps indépendamment de la description structurelle) ;
  - **Les événements** : ils induisent des changements d'état. A ce titre ils sont traités et générés par les composants et les composés ;
  - **Les mesures** : elles permettent d'obtenir des informations sur l'état actuel des composants et des composés. Pour les composés, la mesure minimale est l'accès aux composants ;
  - **Les fonctions de transition** : elles permettent de décrire comment un composant ou un composé réagit aux événements en changeant éventuellement d'état et /ou en engendrant de nouveaux événements.

c) La boite à outils S-Edit

- **Les notions structurelles**

  De façon native la fonctionnalité globale de S-Edit est de permettre la conception et l'animation de diagrammes dont la structure est celle des graphes. Un diagramme est un graphe orienté. La boite à outils du système S-Edit est donc constituée de notions définies par la théorie des graphes. Parce que la théorie des graphes propose un formalisme de représentation graphique des éléments d'un graphe, les notions de la boite à outils S-Edit permettent aussi de décrire les représentions. Le formalisme graphique de la théorie des graphes constitue ainsi le formalisme de base utilisé pour décrire les représentions : une boite à outils graphique en quelque sorte. Ainsi on retrouve les notions suivantes:

- **Le nœud** pour décrire une entité et pour décrire une représentation de cette entité. Pour décrire une représentation de nœud S-Edit mets à disposition un objet graphique ponctuel;
- **L'arc** pour décrire une relation binaire entre entités et pour décrire la représentation de cette relation. Pour décrire une représentation d'arc S-Edit mets à disposition une ligne directe ou brisée;
- **Le graphe** pour décrire un ensemble d'entités en relations.

Un graphe est naturellement asémantique, un diagramme de classe du langage UML est par contre une déclinaison sémantique d'un graphe orienté.

- **Les notions dynamiques**

En ce qui concerne l'aspect dynamique des modèles S-Edit propose les notions d'**action** et de **propriété**. La notion **d'action** permet de décrire un comportement concernant un nœud, un arc ou encore le graphe. Une notion peut disposer de plusieurs actions donc de plusieurs comportements. La notion de **propriété** permet de décrire un attribut variant d'un concept. L'ensemble des valeurs des attributs variants à un instant donné constitue l'état du concept. L'état est modifié lorsque la valeur d'une propriété change.

d) Résultats des jeux : même boite à outils ?

- **Sur l'aspect structurel**

Un modèle d'après la définition implémenté par la boite à outils MIMOSA, est un discours qui articule des concepts. Un modèle est donc un ensemble d'entités dont les articulations constituent des relations binaires. Bref le modèle, tel qu'il est accepté par MIMOSA, est un graphe. Conséquemment, nous pouvons déclarer à ce point qu'en ce qui concerne les notions structurelles qui reflètent la définition d'un modèle les boites à outils de part et d'autre sont identiques. MIMOSA et S-Edit définissent un modèle de la même manière. Nous avons donc les correspondances suivantes :

Tableau 1. *Correspondances des notions de description d'un modèle*

Plateforme MIMOSA	Système S-Edit
Composant	Noeud
Composé	Graphe
Relation intra-composé	Plusieurs arcs

Il faut cependant noter que les nœuds et les arcs, parce qu'ils se situent explicitement dans la théorie des graphes, disposent par rapport aux notions de composants et de relations intra-composés de représentations graphiques. De tel

sorte que les notions de S-Edit peuvent être perçu comme une légère spécialisation des notions de la plateforme MIMOSA.

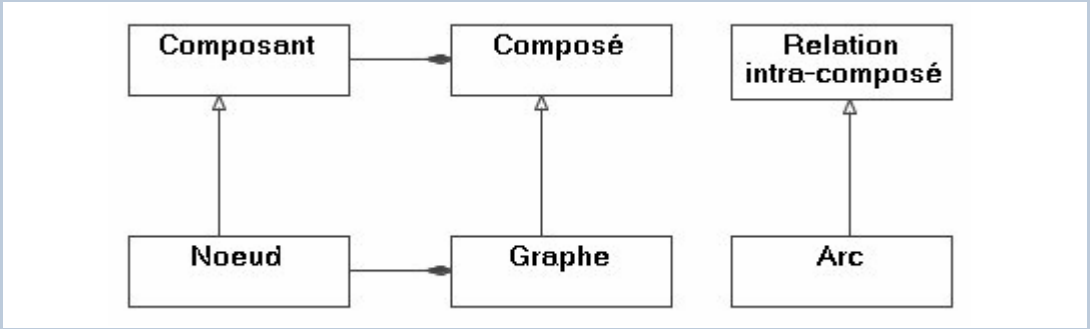


Figure 15. *Légère spécialisation de S-Edit: connotation graphique*

MIMOSA et S-Edit définissent le modèle de la même manière mais contrairement à MIMOSA, S-Edit ne permet pas la modélisation de systèmes complexes car sa boite à outils ne dispose pas de notions permettant de décrire les interactions entres modèles, entres points de vue. En effet S-Edit ne dispose pas de notions équivalentes aux **relations inter-composés** ou aux **relations entre composants et composés**. Donc en ce qui concerne le concept de modèle, dans un point de vue purement abstrait, purement conceptuel nous dirons que S-Edit peut constituer un éditeur de modèles MIMOSA, les modèles étant des déclinaisons de la boite à outils, mais ne peut permettre l'édition d'un système complexe, un système complexe étant un multi modèle.

- **Sur l'aspect dynamique**

  Sur l'aspect dynamique, les deux systèmes sont assez divergents dans les outils qu'ils offrent pour la description des processus. Ainsi MIMOSA propose une unique notion d'état pour décrire les parties des concepts qui évoluent au fil du temps. Alors que S-Edit voit l'état comme étant constitué d'un ensemble de propriété, par conséquent donne directement accès aux propriétés. A proprement parler, Il ne s'agit pas d'une divergence mais plutôt d'une différence de niveau. En effet la notion d'état se trouve à un degré d'abstraction supérieur. L'état vu comme un ensemble de propriété est une spécialisation de la notion d'état proposé par MIMOSA. En terme de comparaison on se retrouve dans le cas de figure suivant :

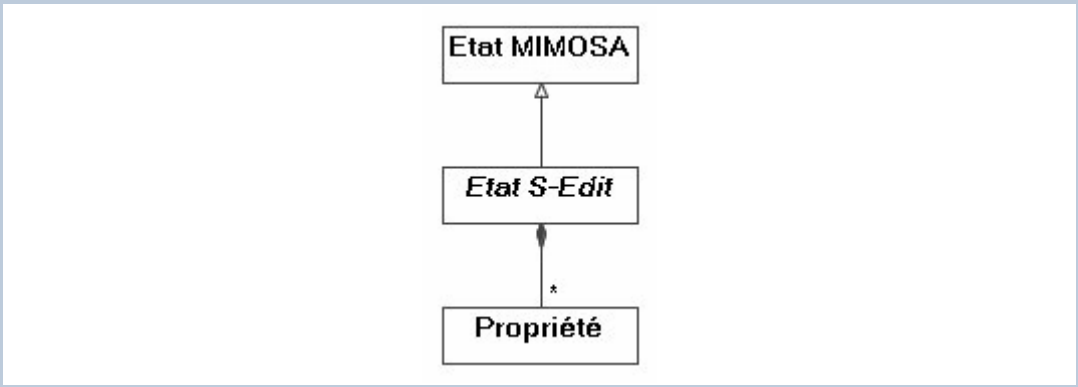


Figure 16. *La notion d'état : degrés d'abstraction différents*

En ce qui concerne les processus qui tendent à modifier les états, MIMOSA propose les notions d'événements et de fonction de transition. Ces notions permettent de décrire le comportement des concepts en fonctions des événements qu'ils reçoivent. A titre d'équivalence S-Edit propose une unique notion d'action qui encapsule à la fois l'événement et la réaction à l'événement.

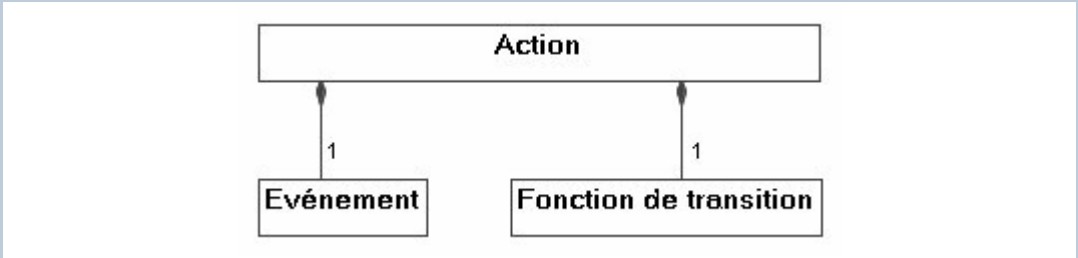


Figure 17. *Comparaison action, événements et fonctions de transition*

Les boîtes à outils sur le plan dynamique ne sont pas identiques, cela implique que S-Edit est, à priori, nativement incapable d'éditer des déclinaisons de la partie dynamique de la boite à outils MIMOSA. Cependant on peut constater que les outils de description de l'aspect dynamique sont identiques aux concepts du diagramme d'état transition d'UML.

2. Implémentations

a) Implémentation de la boite à outils MIMOSA

- **Implémentations des notions structurelles**

  Les classes abstraites et interfaces java implémentant les notions de description d'un modèle ou point de vue sont les suivantes :

  - Les classes permettant d'implémenter la notion de composant :
    - La classe **ComponentType** : elle correspond aux parties de formalismes qui dénotent des concepts catégoriels composants, appelés aussi les types de composants.

- La classe **Component** : elle correspond aux parties des formalismes qui dénotent des concepts individuels unitaires, appelés aussi les instances de composants, ou simplement les composants.

- Les classes permettant d'implémenter la notion de composé :
  - La classe **CompoundType** : elle correspond aux parties des formalismes qui dénotent des concepts catégoriels composés, appelés aussi les types de composés ;
  - La classe **Compound** :elle correspond aux parties des formalismes qui dénotent des concepts individuels composés, appelés aussi les instances de composés, ou simplement les composés.
- L'interface **Name** : les implémentations de cette interface permettent de définir ce qu'est un nom pour un composé.
- Les classes permettant d'implémenter la notion de relation intra composés :
  - la classe **IntraCompoundRelationType** : elle correspond aux parties des formalismes qui dénotent des types de relation entre composants à l'intérieur d'un même composé (relation sur les éléments d'un ensemble) ;
  - la classe **IntraCompoundRelation :** elle implémente l'instance d'un type de relation intra-composés

A l'exception de Component, toutes les classes possèdent un nom (**getName** qui rend un objet de type **Name**). Ce n'est pas le cas des composants parce que nous considérons que le nom n'est rien d'autre que ce qui permet de le désigner dans le composé. On peut dire qu'il en possède un donc mais il n'est pas de même nature. Les concepts individuels se veulent explicitement des instances des concepts catégoriels et mettent donc à disposition le type du concept catégoriel correspondant (**getType**).

Chaque composant définit le moyen d'accéder à son composé (**getCompound**), aux autres composants de son composé (**getComponents(Name)**), ainsi qu'aux composants et composés avec lesquels le composant est mis en relation (**getRelatedComponents(String)**,**getRelatedCompounds(String)**).

Chaque composé permet d'accéder (**getComponents(Name)**), d'ajouter (**addComponent(Name,Component)**), de modifier (**setComponent(Name,Component)**), de détruire (**removeComponents(Name)**) ses composants.

L'implémentation des notions de description de systèmes complexes est construite autour des classes suivantes :

- Les classes permettant d'implémenter la notion de relations inter-composés :
  - La classe **InterCompoundRelationType** : elle correspond aux parties des formalismes qui dénotent des types de relation entre composants de composés distincts.
  - la classe **InterCompoundRelation :** elle implémente l'instance d'un type de relation inter-composés
- Les classes permettant d'implémenter la notion de relation entre un composant et un composé :
  - La classe **ComponentCompoundRelationType** : elle correspond aux parties des formalismes qui dénotent des types de relation entre un composant et un composé (relation entre un élément et un ensemble autre que l'appartenance).
  - la classe **ComponentCompoundRelation :** elle implémente l'instance d'un type de relation entre un composant et un composé.

o **Implémentation des notions dynamiques**

En ce qui concerne l'implémentation des notions dynamiques on retrouve :

- La classe **SimEvent** : elle implémente la notion d'événement ;
- L'interface **MeasureTool** : ses implémentations définissent la notion de mesure, cette interface permet de décrire les instruments de mesures ;
- L'interface **State** : ses implémentions définissent la notion d'état ;
- L'interface **ComponentImplementation** : elle mets à disposition la spécification des événements, des mesures et de leur traitement, pour le cas d'un composant. Cette interface dispose des méthodes :
  - **getIncomingEvents** permettant de définir l'ensemble des événements que le concept accepte ;
  - **getMeasureTools** permettant de définir l'ensemble des mesures que l'on peut appliquer sur le concept ;
  - **handleEvent** permettant de définir la fonction de transition qui reçoit un événement et le traite en fonction de l'état interne ;

- **handleMeasure** permettant de définir la fonction de traitement de la mesure, elle reçoit la spécification d'une mesure et rend le résultat de la mesure ;

- L'interface **CompoundImplementation** : elle mets à disposition la spécification des événements, des mesures et de leur traitement, pour le cas d'un composé. Cette interface dispose des méthodes :

  - **getIncomingEvents** permettant de définir l'ensemble des événements que le concept accepte ;
  - **getMeasureTools** permettant de définir l'ensemble des mesures que l'on peut appliquer sur le concept ;
  - **handleEvent** permettant de définir la fonction de transition qui reçoit un événement et le traite en fonction de l'état interne ;
  - **handleMeasure** permettant de définir la fonction de traitement de la mesure, elle reçoit la spécification d'une mesure et rend le résultat de la mesure.

o **Modélisation de la boite à outils d'après le formalisme du diagramme de classe d'UML**

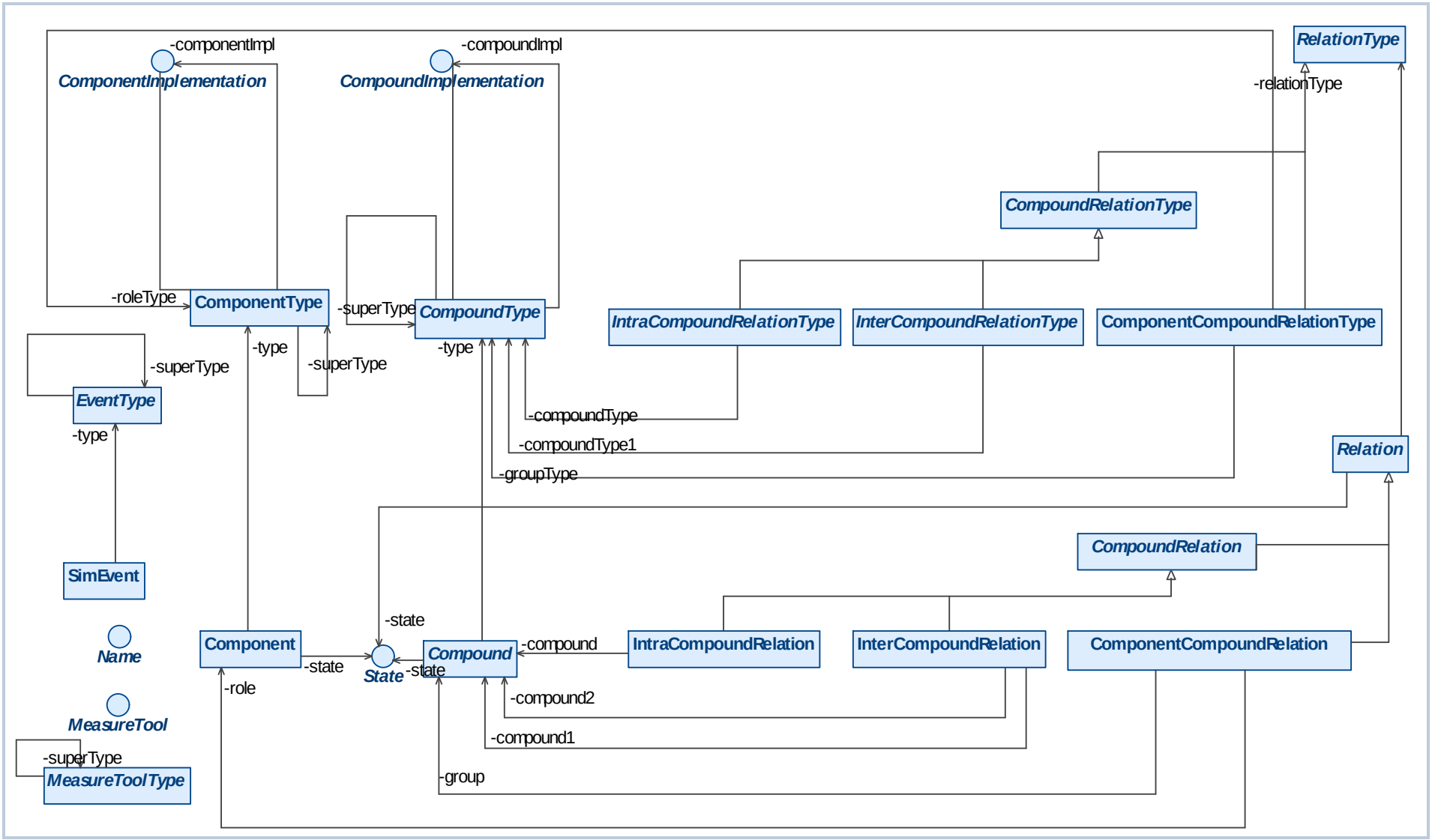


Figure 18. *La boite à outils de la plateforme MIMOSA*

b) Implémentation de la boite à outils S-Edit

- **Implémentations des notions structurelles**

  La boite à outils de S-Edit au niveau structurelle est une implémentation des notions de la théorie des graphes. Le graphe est implémenté par la classe **Formalism** et la classe **Structure**. La classe **Formalism** définit le formalisme de base des catégories, des types de graphes. La classe **Structure** définit le formalisme de base des instances de graphes dont les types ont été définis par les sous classes de la classe **Formalism**. S-Edit considère dans l'implémentation du graphe qu'un graphe est un ensemble d'éléments, que ces éléments soient des nœuds ou des arcs. Chaque élément dispose d'une représentation graphique qu'il s'agisse d'un objet graphique ponctuel ou linéaire. Les classes abstraites **ElementDesc**, **SElement** et **GObject** implémentent la notion d'élément :

  - La classe **ElementDesc** implémente la notion de catégorie d'élément. Cette classe définit le formalisme d'un type d'élément, elle permet la description d'un élément ainsi que la description de la représentation graphique de l'élément ;
  - La classe **SElement** implémente la notion de d'instance de catégories d'éléments ;
  - La classe **GObject** implémente la notion d'instance d'objet graphique de visualisation. La catégorie d'objet graphique est implémentée par la classe **ElementDesc**. Un objet graphique est un concept d'un modèle géométrique. La classe **GObject** de pair avec la classe **ElementDesc** définit l'abstraction utilitaire de base des concepts de ce modèle. A ce titre la classe **GObject** doit être considéré comme un outil graphique. Une instance de la classe **SElement**, instance au sens Java, est liée à une instance de la classe **GObject**.

  Ces trois classes définissent de façon basique la notion d'élément, éléments d'un graphe. Les éléments d'un graphe peuvent soit être des nœuds, soit être des arcs. Un nœud est un élément disposant de spécificités particulières, le nœud doit être lié à un objet graphique ponctuel. Les classes **NodeDesc**, **SNode** et **GNode** implémentent la notion de nœud :

  - La classe **NodeDesc** définit le formalisme de base des catégories, des types de nœuds.

- La classe abstraite **SNode** définit le formalisme de base des instances de nœuds.
- La classe abstraite **GNode** définit le formalisme de base des objets graphique ponctuels de représentation de nœuds.

De la même manière les arcs sont des éléments spécifiques dont l'implémentation est assurée par les classes **ArrowDesc**, **SArrow** et **GArrow** :

- La classe **ArrowDesc** définit le formalisme de base des catégories, des types d'arcs.
- La classe **SArrow** définit le formalisme de base des instances d'arcs.
- La classe **GArrow** définit le formalisme de base des objets graphique linéaires.

o **Implémentations des notions dynamiques**

Au niveau dynamique la notion d'**action** est implémentée par une méthode java publique sans arguments retournant void et une instance de la classe **ActionDesc** permettant de référencer la méthode java. Les actions sont propres aux concepts dont elles décrivent le comportement, Aussi les méthodes java cibles doivent figurer dans les classes qui définissent le formalisme des concepts. Ainsi une déclinaison de la classe SNode servant à modéliser une vache devrait disposer d'une méthode : « *public void manger ()* » qui sera référencée sous le label « Manger herbe » avec une icône représentant un gâteau … hum par exemple. Ainsi dans ce cas la méthode java publique sans arguments retournant void implémente le formalisme de base de description d'actions.

Pareillement la notion de **propriété** est implémentée par une paire de méthodes « getter/setter » publiques généralement associée à un attribut de la classe définissant le concept. Ainsi par défaut tout élément dispose de la propriété **Label** qui est implémenté par les méthodes publiques **getLabel()** et **setLabel()**, ces méthodes sont associé à l'attribut **label**;

- **Modélisation de la boite à outils S-Edit d'après le formalisme du diagramme de classe d'UML**

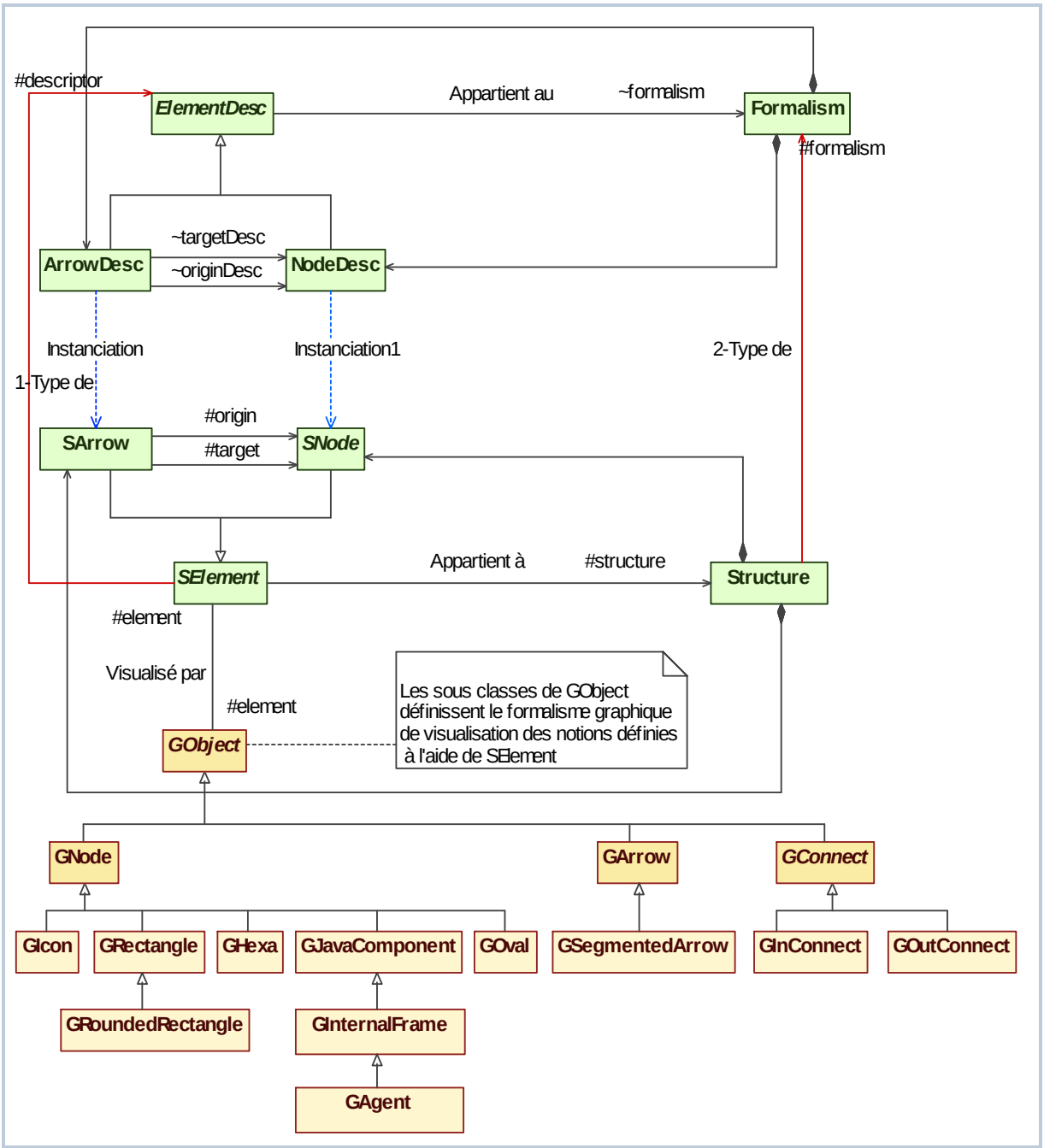


Figure 19. *La boite à outils S-Edit*

3. Sauvegarde des modèles

Qu'il s'agisse de MIMOSA ou de S-Edit la sauvegarde des modèles se fait dans le format XML. Ce choix est motivé de part et d'autre par les capacités de description du langage XML. Les fichiers XML s'ils servent de sauvegarde, n'en constituent pas moins des interfaces utilisateur de description des modèles. Ces capacités sont pleinement utilisées par S-Edit qui s'en sert pour décrire les catégories d'objets du modèle. Cependant, le

langage XML bien qu'assez simple d'accès demeure assez rébarbatif pour les non informaticiens. Pour palier à cet inconvénient d'ordre « ergonomique » MIMOSA propose une interface graphique générique de description de catégories.

a) Sauvegarde des modèles MIMOSA

Chaque notion de la boite à outils MIMOSA, dans son implémentation dispose d'une méthode **toXML()**. Cette méthode dans sa définition de base sauvegarde au format XML l'état de la notion. Cette méthode est surdéfinie dés lors que la notion devient sémantique. Etant sémantique la description XML de la notion doit en refléter la spécificité. Il est évident que l'on retrouve une nouvelle implémentation de la boite à outils mais cette fois à travers le langage XML.

b) Sauvegarde des modèles S-Edit

Au niveau de S-Edit en ce qui concerne la sauvegarde il faut comprendre que le format XML est utilisé au même titre qu'une interface graphique pour décrire les catégories d'objets du discours. Cela implique que les concepts catégoriels ne font pas réellement l'objet de sauvegarde car ils sont décrits de façon statique à travers un fichier XML. Le modèle concret par contre fait l'objet de sauvegarde car il est construit pendant l'exécution par l'éditeur de diagrammes. Mais la démarche de sauvegarde manque de généricité car elle n'inclut pas de possibilité de surdéfinition ; la sauvegarde est plutôt géré par deux classes : la classe **XMLStructureSaver** chargé de sauvegarder un modèle et tout ce qu'il contient (nœuds, arcs, connecteurs) et la classe **XMLStructureLoader** qui permet de charger les modèles. Ce manque de généricité est compensé par la présence des propriétés, en effet les propriétés disposent d'une description XML il s'agit donc d'assimiler la spécialisation des concepts à l'adjonction de nouvelles propriétés. Définir un nouveau type c'est définir un nœud avec de nouvelles propriétés.

c) Comparaison

A titre de comparaison on retiendra seulement que le système S-Edit effectue une description statique des catégories du modèle alors que la plateforme MIMOSA effectue une description interactive des catégories du modèle. Etant interactive cette description se doit d'être sauvegardé dans le format XML au même titre que les concepts individuels.

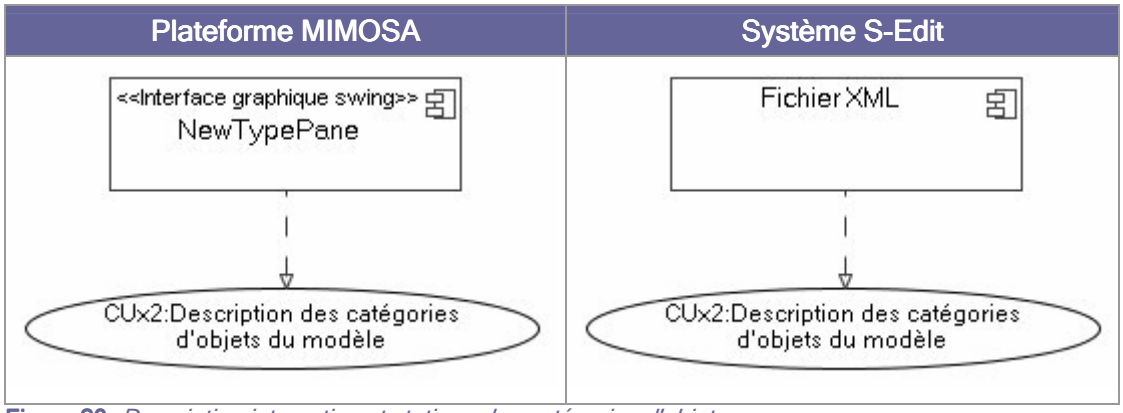


Figure 20. *Description interactive et statique des catégories d'objets*

4. Bilan

Nous avons vu au cours de chapitre que les notions structurelles qui permettent de décrire les modèles au niveau de S-Edit comme au niveau de MIMOSA étaient identiques. Les notions de S-Edit étant légèrement plus sémantique car introduisant un aspect graphique. Aussi nous pouvons d'ors et déjà répondre par l'affirmative à la question « S-Edit peut-il constituer un éditeur de modèle MIMOSA ? ». Le problème qui se pose maintenant réside au niveau des implémentations. En effet nous disposons de deux implémentations pour des notions identiques. Les correspondances suivantes ont pu être établies :

Tableau 2. *Correspondances des implémentations*

Concepts	Implémentation MIMOSA	Implémentation S-Edit
Concept individuel composant	Component	SNode
Concept individuel composé	Compound	Structure
Relation individuelle intra composé	IntraCompoundRelation	SArrow
Concept catégoriel composant	ComponentType	NodeDesc
Concept catégoriel composé	CompoundType	Formalism
Relation catégorielle intra composé	IntraCompoundRelationType	ArrowDesc

Dans la suite de notre travail nous allons déterminer comment fusionner ces deux implémentations. Les considérations sur le format XML et autres descriptions techniques nous serons d'une grande utilité dans le chapitre suivant.

Chapitre IV.
Edition de modèles MIMOSA

A.Assimilations

Dans ce chapitre nous allons proposer une solution au problème qui nous a été posé. Le problème était le suivant : S-Edit est un système qui associe aux concepts des modèles une représentation graphiques de tel sorte que les modèles sont construits sous la forme de diagrammes. Il s'agissait d'utiliser S-Edit pour construire des modèles MIMOSA. Nous avons vu que la difficulté majeure de cette opération résidait dans le fait que S-Edit en plus de présenter des concepts graphiques, présentait aussi des concepts de définitions de modèles. Créant ainsi une redondance avec la boite à outils de la plateforme MIMOSA. Pour palier à cet inconvénient, l'on a vu qu'il devait y avoir une assimilation entre les deux boites à outils.

Nous allons débuter la résolution du problème par la description de ce qu'il faut entendre par assimilation. L'assimilation ne doit en aucun cas signifier la destruction d'une implémentation. Elle ne doit pas signifier la destruction de l'implémentation MIMOSA car sinon cet exposé n'aurait plus lieu d'être ; elle ne doit pas non plus signifier la destruction de l'implémentation S-Edit car un couplage, en plus de la migration de fonctionnalités, a pour intérêt de profiter de l'expertise évolutive des concepteurs du système couplé. Un couplage doit être compatible avec toutes les versions d'un logiciel. Par destruction nous entendons par exemple la suppression de la boite à outils S-Edit et l'adaptation du contrôleur (sens MVC) à la boite à outils MIMOSA. Il faut plutôt que l'instanciation d'un nœud S-Edit entraîne l'instanciation d'un composant MIMOSA et qu'il en soit de même pour les arcs et les relations, pour les graphes et les composés. Nous avons constaté au niveau de l'analyse que la boite à outils MIMOSA est une généralisation de la boite à outils S-Edit. Pour réaliser cette assimilation nous n'aurons donc qu'à réaliser la relation de généralisation que nous avons détectés.

Au niveau de la programmation orienté objet pour réaliser une relation de généralisation il est nécessaire de mettre en place un mécanisme de propagation des attributs et opérations des super classes vers les sous classes. Ce mécanisme est réalisé de deux manières. La première est celle des langages de programmation objet et de la technologie de compilation sous-jacente qui implémentent le mécanisme et proposent le mécanisme d'héritage. La deuxième manière est celle du programmeur qui réalise la

généralisation par délégation en agrégeant les super classes dans les sous classes. Chacune de ces deux techniques disposent d'avantages comme d'inconvénients. En ce qui concerne le mécanisme d'héritage il permet une propagation automatique, pour ne pas dire naturelle, des attributs et des méthodes ; mais elle établit un couplage fort entre les classes, ne permettant pas ainsi à une classe de changer de responsabilité durant l'exécution. L'héritage établit une classification statique et par conséquent rigide. La délégation, par contre, même si elle impose une propagation manuelle des propriétés, permet une classification dynamique qui apporte une plus grande flexibilité. La question qui se pose à présent est : devrait on mettre en oeuvre une assimilation statique ou dynamique entre les boites à outils MIMOSA et S-Edit. Quelles sont les conséquences de chaque approche ?

B.Assimilation par héritage et interfaçage

1. Description

L'assimilation par héritage part du constat que la boite à outils S-Edit est une spécialisation légèrement plus sémantique de la boite à outils MIMOSA. D'où l'idée de fusionner les boites à outils à travers le mécanisme d'héritage. Dans la philosophie de la plateforme cette opération s'apparente à une création de formalisme, en l'occurrence un formalisme S-Edit. Cependant les concepts de ce formalisme ne constituent au final qu'une autre boite à outils car, asémantique, qui n'aura d'autre particularité que d'être graphique. Pour les puristes cette approche se trouve être la plus élégante car elle fusionne des éléments conceptuellement identiques. Un nœud se trouve être un composant, un arc se trouve être une relation intra composé et le graphe se trouve être un composé. Ainsi les notions basiques de MIMOSA se retrouvent enrichis de méta données permettant leurs affichage et édition graphique. L'héritage par ailleurs, contrairement à la réécriture, n'entraîne pas l'adjonction directe des méta données aux classes MIMOSA. Ces méta données peuvent être considérées comme encombrantes surtout lorsque l'on développe des formalismes qui ne nécessitent pas de représentations graphiques.

2. Implémentation

En ce qui concerne l'implémentation, la situation est moins idéale. Ce que l'on aimerait nous c'est placer immédiatement **SNode** en tant que sous classe de **Component**, **SArrow** en tant que sous classe de **IntraCompoundRelation** et de même pour les concepts catégoriels de nœud et d'arc. Mais les classes **SElement** au niveau individuel et **ElementDesc** au niveau catégoriel nous font obstacles car elles généralisent l'ensemble des notions individuelles et catégorielles de nœud et d'arc – les nœuds et les arcs sont des éléments. Solution ? Nous ne pouvons pas supprimer ces classes car le contrôleur S-Edit (toujours au sens MVC) y fait référence. Nous n'avons donc qu'une seule solution transformer ces classes en interfaces et introduire de nouvelles classes pour implémenter ces

interfaces. Nous utilisons de nouvelles classes car l'on ne veut pas modifier l'implémentation des classes S-Edit. On veut demeurer dans une optique de couplage même si l'on vient de violer de manière imperceptible (pour S-Edit) ce principe. Dans cet optique on introduit aussi des classes intermédiaires entre **CompoundType** et **Formalism** et entre Compound et Structure, même s'il n'y a pas d'obstacle à ce niveau. En résumé les classes et interfaces que nous allons introduire sont les suivantes :

- **L'interface ElementDesc** : elle se substitue à la classe du même nom définie par S-Edit.
- **L'interface SElement** : elle se substitue à la classe du même nom définie par S-Edit.
- **La classe SEditComponentType**: cette classe implémente l'interface **ElementDesc** et hérite de la classe MIMOSA **ComponentType**. De cette classe hérite la classe **NodeDesc**.
- **La classe SEditIntraCompoundRelationType** : cette classe implémente l'interface **ElementDesc** et hérite de la classe MIMOSA **IntraCompoundRelationType**. De cette classe hérite la classe **ArrowDesc**.
- **La classe SEditCompoundType** : cette classe sert uniquement à implémenter les méthodes abstraites des super classes MIMOSA dont elle hérite : elle hérite de la classe MIMOSA **CompoundType**. De cette classe hérite la classe **Formalism**.
- **La classe SEditComponent** : cette classe implémente l'interface **SElement** et hérite de la classe MIMOSA **Component**. De cette classe hérite la classe **SNode**.
- **La classe SEditIntraCompoundRelation** : cette classe implémente l'interface **SElement** et hérite de la classe MIMOSA **IntraCompoundRelation**. De cette classe hérite la classe **SArrow**.
- **La classe SEditCompound** : cette classe sert uniquement à implémenter les méthodes abstraites des super classes MIMOSA dont elle hérite : elle hérite de la classe MIMOSA **Compound**. De cette classe hérite la classe **Structure**.

Le diagramme de classe suivant illustre la boite à outils unifiée du système résultant :

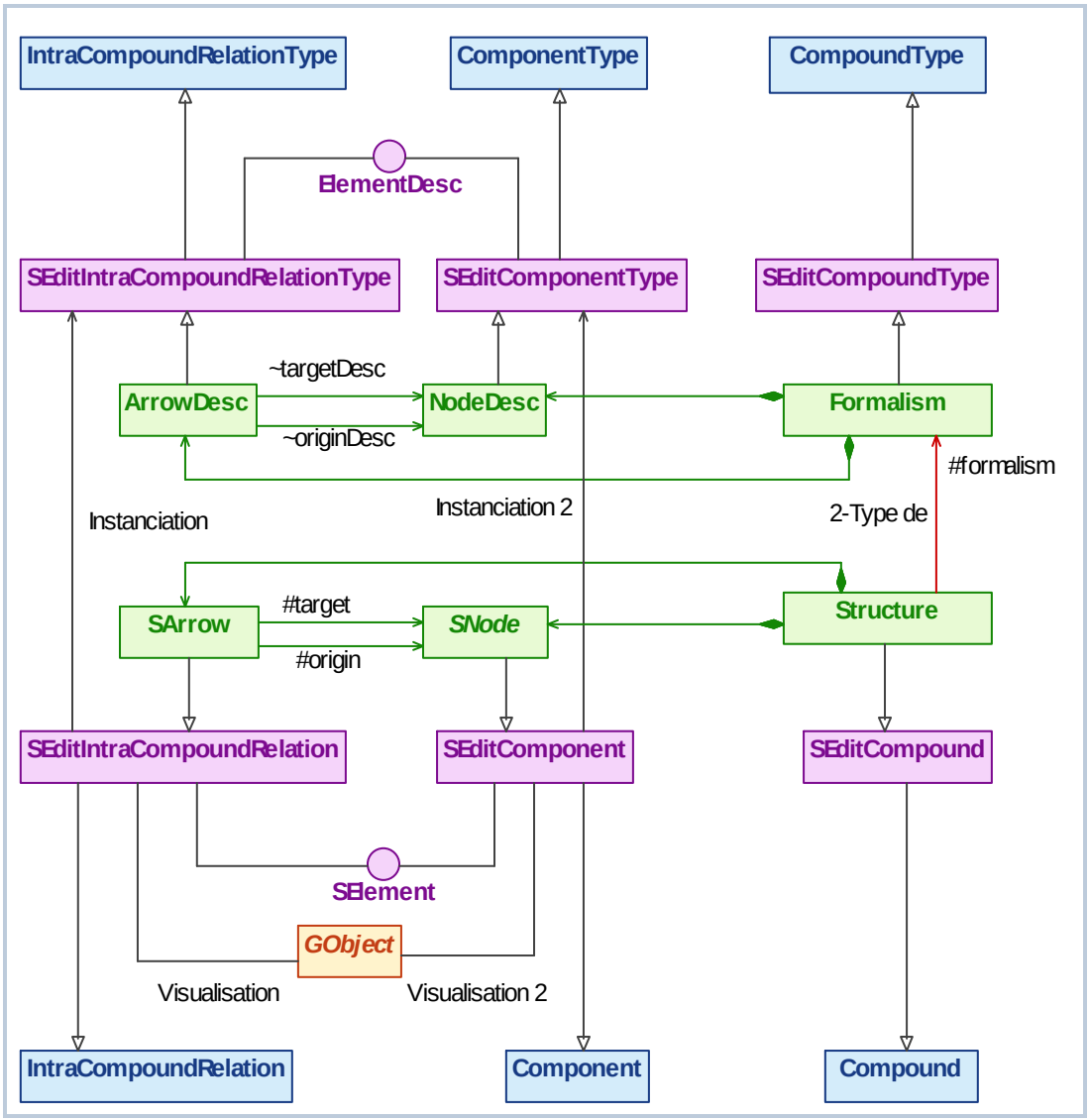


Figure 21. *Une boite à outils unifiée*

Classes MIMOSA
Classes S-Edit
Classes de couplage

3. Dépendances

En enrichissant la boite à outils S-Edit nous en avons aussi défini une spécialisation. Or la spécialisation dans la philosophie de la plateforme est une création de formalisme. Nous avons donc défini un nouveau formalisme. Les formalismes sont caractérisés par des définitions de catégories et d'individus. MIMOSA étant une plateforme générique, ces nouvelles catégories et ces nouveaux individus ne sont connus que de par leurs créateurs. Ces créateurs, en l'occurrence nous, doivent informer la plateforme sur la façon d'instancier et d'initialiser les éléments contenu dans ce formalisme. La généricité se traduit toujours par la définition d'une abstraction de base. Ainsi la plateforme nous impose de présenter une interface graphique de création et de description des catégories contenu dans notre formalisme. Pour ce faire elle met à notre disposition, comme support

à la généricité, une classe abstraite **NewTypePane** qui fournit une interface graphique de base pour l'instanciation des types. Il s'agira pour nous de définir des sous classes de **NewTypePane** implémentant les spécificités de description des types propres à S-Edit. Cette méthodologie de description des types, des catégories est implémenté – si l'on peut dire – par la DTD des fichier XML de description de formalismes S-Edit (*voir annexe A*). Notre travail consiste à définir un composant swing (interface graphique java) qui va implémenter à nouveau cette méthodologie. Le format de description XML des catégories va constituer le format de sauvegarde des catégories du formalisme S-Edit.

En ce qui concerne l'instanciation et l'initialisation des concepts individuels à proprement parler la construction des modèles ou encore l'édition des modèles nous disposons bien sûr de l'éditeur de diagramme du système S-Edit. Pour son intégration dans la plateforme MIMOSA, la classe **CompoundView** nous est proposée. Cette classe définit une interface graphique de base pour l'édition de modèles. Nous en définissons donc une sous classe que l'on nomme Editor; la classe **Editor** va encapsuler l'éditeur de diagramme S-Edit. Dans notre implémentation, pour le rôle le l'éditeur S-Edit nous avons choisi la classe **FormalismStructureAgent**. Cette classe conformément à l'architecture définie dans la plateforme Madkit implémente un agent qui à partir d'un formalisme génère un éditeur de diagramme approprié. Pour des raisons de paramétrage nous avons préféré cette classe à la classe qui implémente directement l'éditeur de diagramme : l'agent est un expert dans la génération d'éditeurs de diagrammes. En résumé, parmi les classes qui définissent des interfaces utilisateurs de description et de construction des modèles on compte :

- La classe abstraite **NewElementDescPane** : cette classe défini le protocole de base de description d'un élément. On a vu précédemment qu'un nœud et un arc sont considéré comme étant les éléments d'un graphe. L'interface graphique définie par cette classe permet de décrire les propriétés communes aux nœuds et aux arcs.
- La classe **NewNodeDescPane** : elle définit le protocole de base de description de nœuds. La description est contenue dans une instance de la classe **NodeDesc**.
- La classe **NewArrowDescPane** : elle définit le protocole de base de description d'arcs. La description est contenue dans une instance de la classe **ArrowDesc**.
- La classe **NewFormalismPane** : elle définit le protocole de base de description du formalisme d'un graphe. L'interface graphique permet de sélectionner les types nœuds et types d'arcs du formalisme. La description est contenue dans une instance de la classe **Formalism**.
- La classe **Editor** : elle encapsule l'éditeur de diagramme S-Edit.

Dans un couplage, ce que l'on recherche en plus de la simple migration des fonctionnalités, c'est l'expertise des concepteurs du système couplé. Alors qu'une version d'un logiciel est figée dans le

temps, les concepteurs de logiciel sont en évolution permanente, ils sont perpétuellement entrain de penser aux manières d'améliorer leurs logiciels, tant au niveau de la qualité de l'implémentation des fonctionnalités, qu'au niveau de leurs nombre. Et ultimement de façon répétitive, produit de leurs recherches, de leurs expertises grandissantes, une nouvelle version du logiciel est produite. Un bon couplage doit prendre en compte ce facteur. Il doit être conçu de façon à être compatible avec plusieurs versions d'un même logiciel. Il ne pourra cependant pas intégrer totalement ce facteur car certaines versions d'un même logiciel n'ont de commun que le nom. Notre manière que nous avons eu de prendre en compte ce facteur est l'emploi de la réflexivité. En effet la plus part des classes de description des catégories que nous avons conçues génèrent l'interface graphique utilisateur en fonctions des propriétés (telles que nous les avions définies dans le chapitre précédent) décelées sur les classes du modèle. Etant dans une plateforme générique, où virtuellement tout peut être redéfini, cette méthode peut sembler de prime abords assez restrictive car elle impose aux programmeurs une certaine rigueur dans le codage ; mais elle doit plutôt être perçue comme un outil de généricité supplémentaire. En effet au niveau de l'interface graphique, des espaces sont aménagés pour servir les extensions futures ; mais les concepteurs de ces futures extensions peuvent, s'ils le veulent, s'alléger de la tache de création de nouvelles interfaces en structurant leurs code de sorte à laisser le système faire le travail. Nous avons ainsi défini à travers la classe **PropertyEditor** un éditeur de propriété. L'éditeur de propriété utilise un filtre défini par la classe **PropertyFilter** lui permettant de d'extraire, dans la définition d'une classe, les propriétés. La classe **PropertyFilter** implémente l'interface **MethodFilter**. L'interface **MethodFilter** est le produit de l'abstraction entre une propriété et une action, ces deux notions nécessitent une extraction de méthodes au niveau des classes mais avec des critères différents. Ainsi contrairement à une propriété une action correspond toujours à une méthode au lieu de deux. Un filtre de propriétés extrais les propriétés et les stockent dans une instance de la classe **PropertyTableModel**.

Selon le pattern MVC, le sous-système constitué par ces classes, constitue le contrôleur de l'applicatif d'édition de modèles. Le diagramme suivant schématise ces classes.

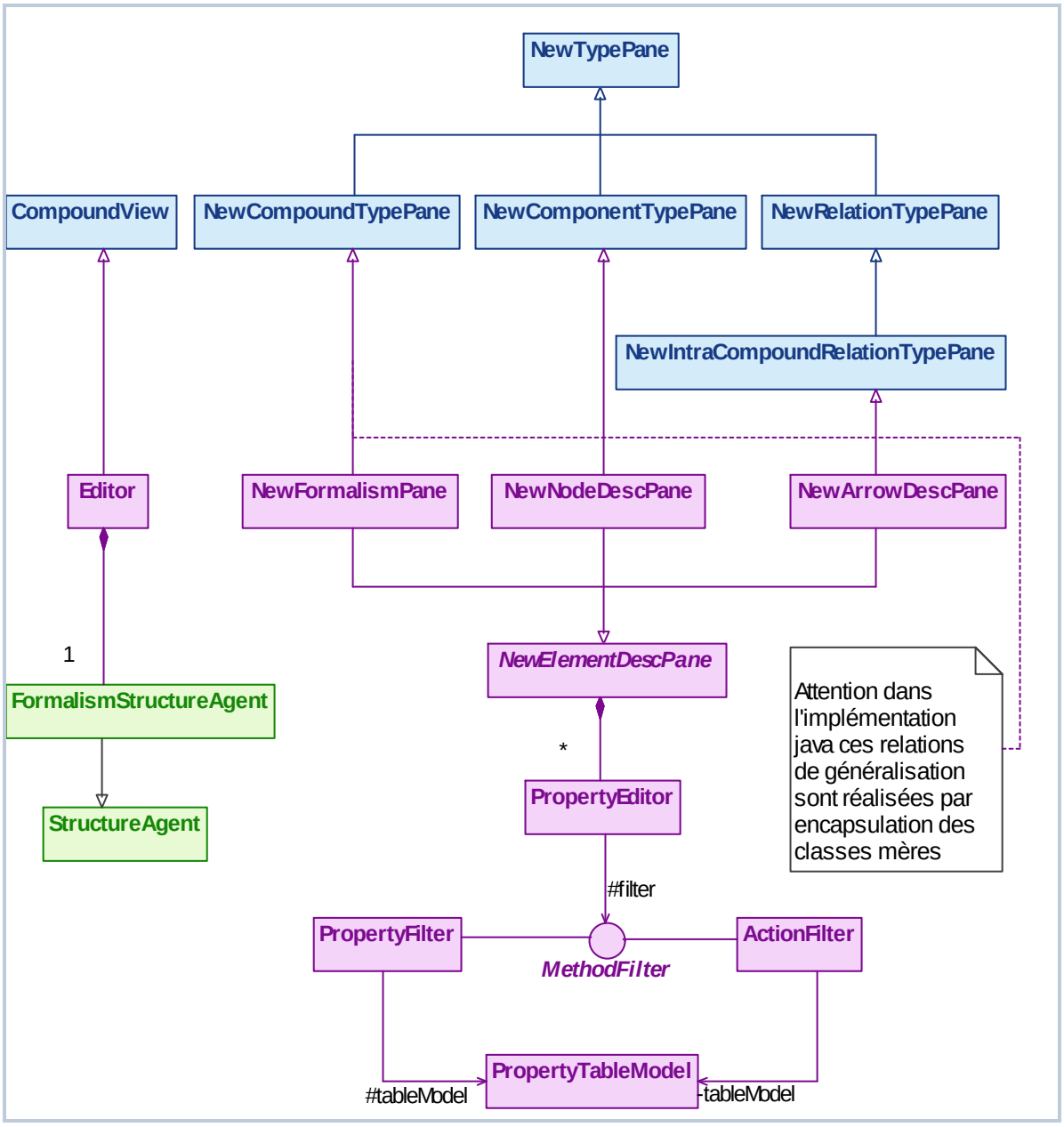
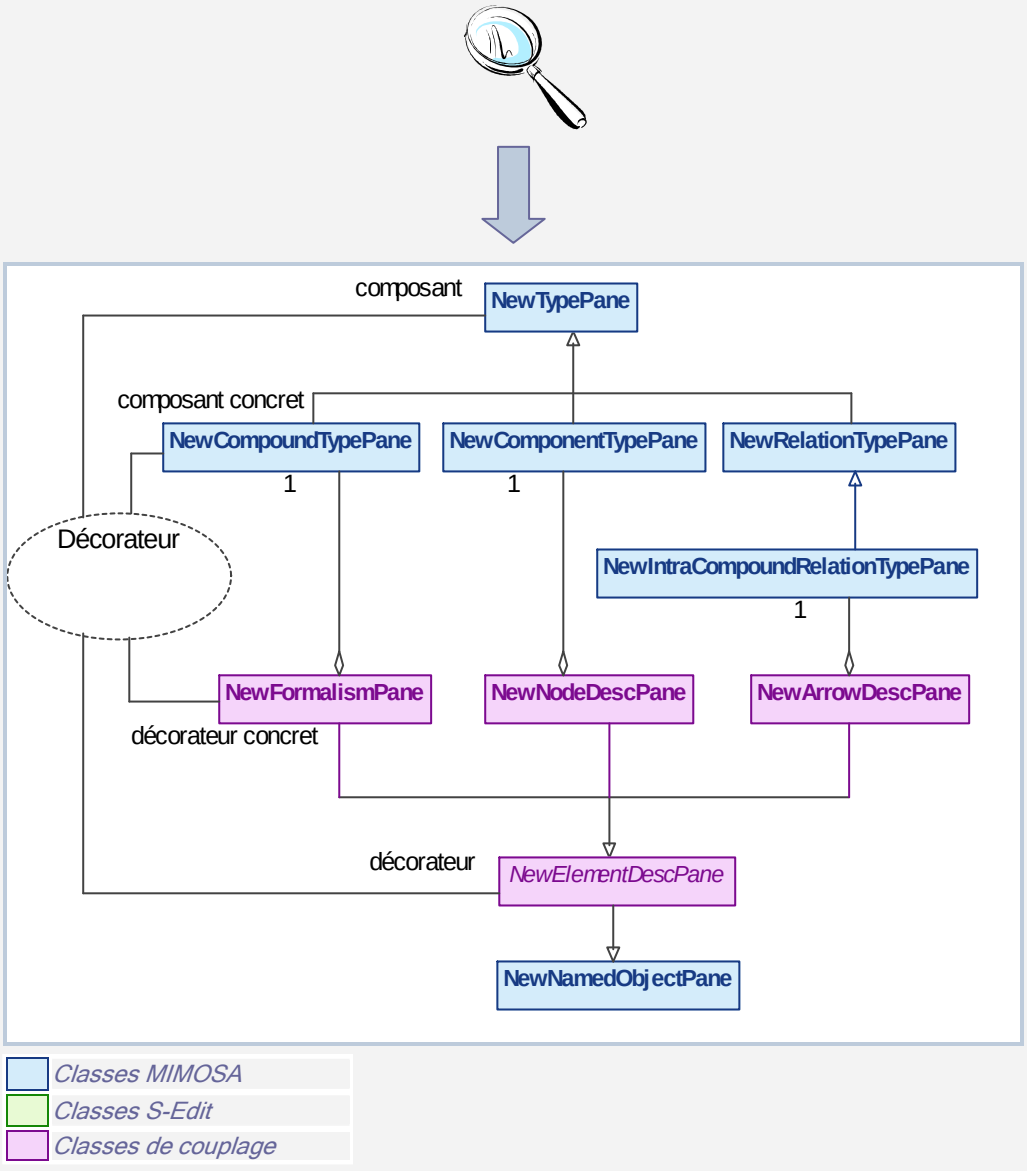

NewTypePane
CompoundView
NewCompoundTypePane
NewComponentTypePane
NewRelationTypePane
NewIntraCompoundRelationTypePane
Editor
NewFormalismPane
NewNodeDescPane
NewArrowDescPane
1
NewElementDescPane
FormalismStructureAgent
*
PropertyEditor
StructureAgent
Attention dans l'implémentation java ces relations de généralisation sont réalisées par encapsulation des classes mères
#filter
PropertyFilter
ActionFilter
MethodFilter
PropertyTableModel
#tableModel
-tableModel

Figure 22.*Contrôleur (MVC)*

*Dans un point de vue purement conceptuel, les classes **NewFormalismPane**, **NewNodeDescPane**, **NewArrowDescPane** sont des sous classes de **NewTypePane**. Mais les nœuds et les arcs dans leur protocole de description présentant de fortes similitudes nous avons décidé de généraliser ces similitudes dans la classe abstraite **NewElementDescPane**. Java ne permettant pas la généralisation multiple nous avons eu recours au pattern décorateur pour simuler un héritage en encapsulant les classes **NewXTypePane**. Ensuite pour que nos classes soient perceptibles de MIMOSA nous avons fait hériter **NewElementDesc** de la classe **NewNamedObjectPane** qui généralise le concept d'interface de création d'entités nommées dans la plateforme. Nous aurions pu opter pour l'option inverse et encapsuler **NewElementDescPane** à la place mais les layouts des classes MIMOSA nous auraient trop contraints. Le diagramme suivant illustre donc l'implémentation de cet héritage multiple en java.*



4. Protocole de description : création de catégories S-Edit

Le format XML nous présente un protocole statique de description des catégories d'objet du discours. Nous nous en sommes inspiré pour mettre en place un protocole interactif de description dont les grandes lignes sont les suivantes. Une description, une catégorie est stockée en mémoire par un objet de type **ElementDesc** pour ce qui est des éléments du graphe ou par un objet de type

Formalism s'il s'agit de la description du graphe. Un protocole de description permet d'initialiser cet objet. Pour ce faire le protocole qui est implémenté par un composant swing va d'abords présenter au modélisateur un éditeur de propriété lui permettant d'initialiser les propriétés de base de la catégorie, généralement ces propriétés correspondent aux attributs de l'objet. Parmi ces attributs de base où l'on retrouve le libellé, la description, l'icône associée, les attributs **elementClass** et **graphicClass** sont les plus importants ; car ils permettent respectivement à travers la sélection d'une classe java de sélectionner le formalisme de l'instance (au sens modélisation) de cette catégorie et de sélectionner la forme graphique symbolisant l'instance. Ainsi Une instance de la catégorie en cours de création pourra soit être un état du formalisme UML des diagrammes d'état transition ou soit être une transition des réseaux de Pétri. Une fois que le protocole est renseigné sur la classe des instances, il crée deux éditeurs. Le premier, un éditeur de propriété, détecte les propriétés sur la classe et permet au modélisateur de sélectionner et d'initialiser les propriétés pertinentes à la notion qu'il est entrain de décrire. A ce moment la classe instance joue plutôt le rôle de dictionnaire de propriétés. Dictionnaire de propriétés mais aussi d'actions. En effet le deuxième éditeur, permet de sélectionner parmi les actions détectées sur la classe celles qui sont pertinentes à son modèle. L'éditeur d'action lui permet alors d'associer à chaque action un libellé.

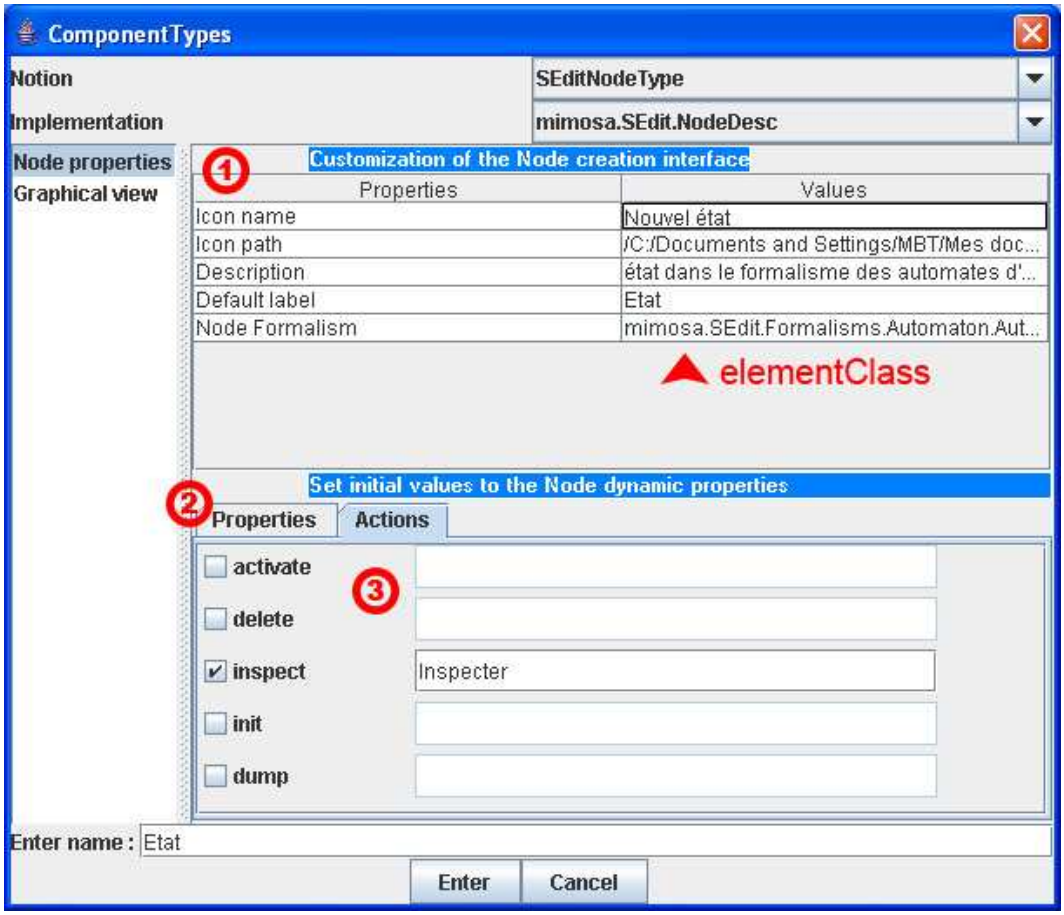


Figure 23. *Editeurs*

1 *Editeur des propriétés de base de la catégorie*

② *Editeur des propriétés de l'instance*
③ *Editeur des actions de l'instance*

Après ces trois interactions du modélisateur Il ne reste plus qu'à décrire graphiquement la notion. Un quatrième éditeur permet la sélection et l'initialisation des propriétés graphiques. Une catégorie étant une composition de ces quatre types de propriétés, il ne reste plus qu'à instancier **ElementDesc** ou **Formalism** pour disposer d'une catégorie.

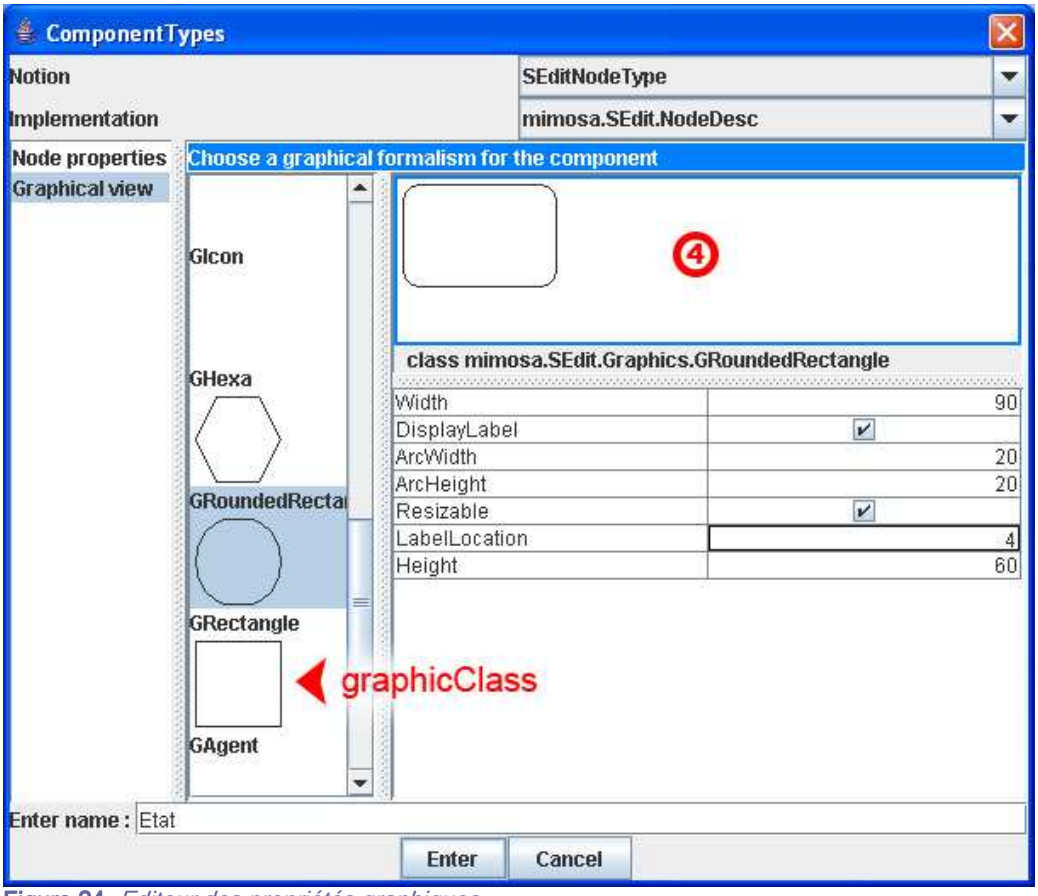


Figure 24. *Editeur des propriétés graphiques*

④ *Editeur des propriétés de l'objet graphique associé aux instances*

La description des catégories de composés diffère légèrement en ce sens que le graphe en tant que composé ne dispose pas de propriétés graphiques par contre on retrouve au niveau des propriétés de base la liste des nœuds. Par ailleurs l'attribut qui permet la sélection de la classe des instances est nommé **structureClass**.

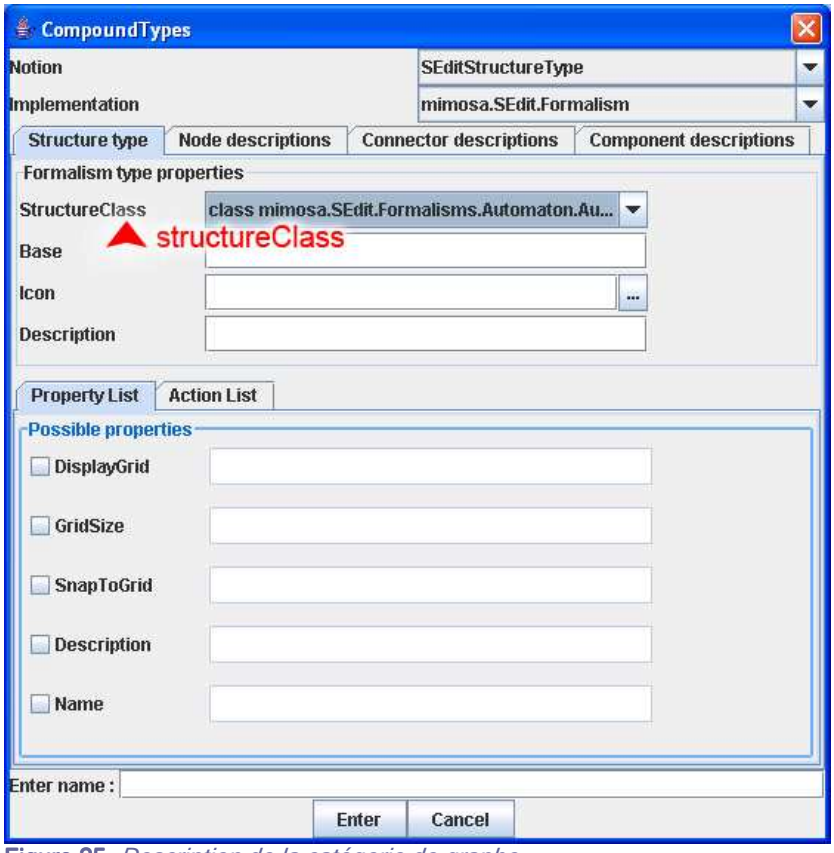


Figure 25. *Description de la catégorie de graphe*

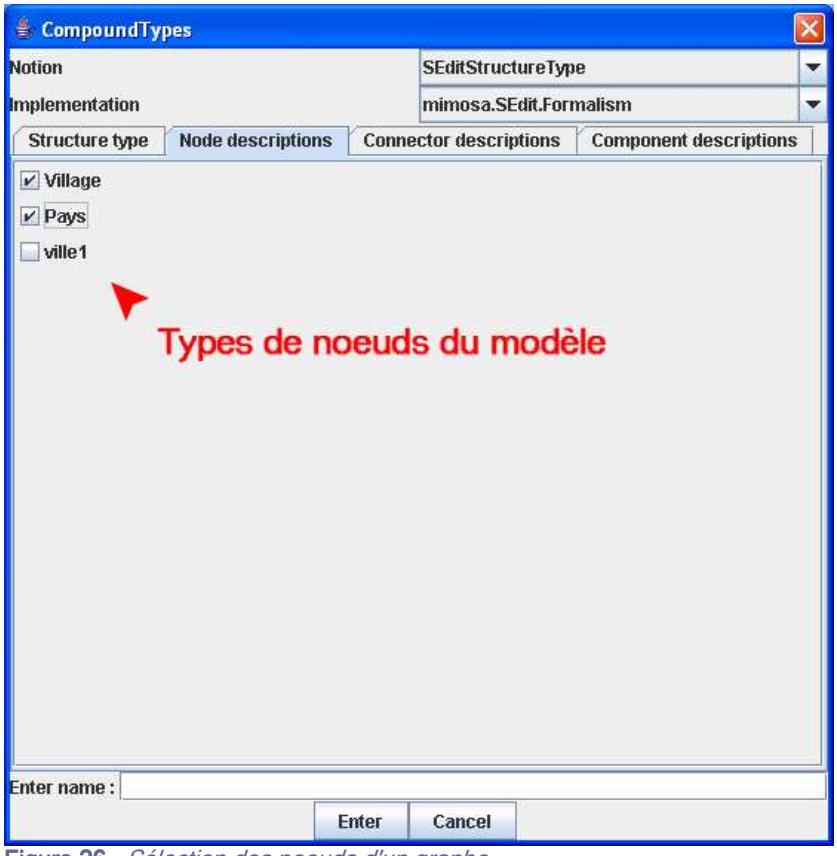


Figure 26. *Sélection des noeuds d'un graphe*

Le diagramme de séquence suivant résume l'ensemble du protocole que nous venons de décrire.

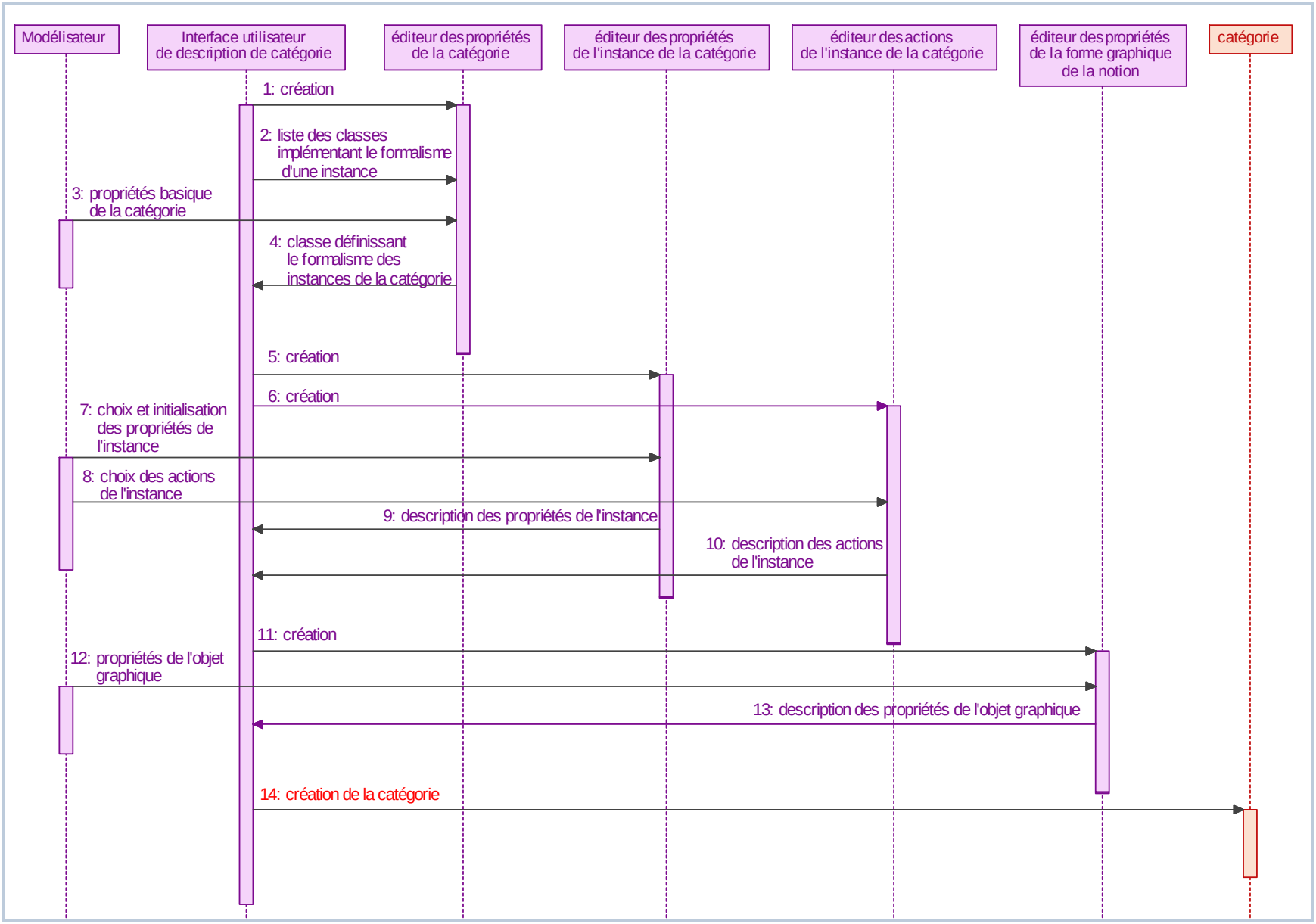


Figure 27. *Le protocole de description des catégories*

5. Sauvegarde XML

Maintenant que le protocole de description est implémenté de façon interactive, le protocole statique implémenté par le format XML va être utilisé pour la sauvegarde des modèles S-Edit. De primes abords on est tenté d'utiliser à cet effet les classes définies par S-Edit mais on se heurte à la difficulté suivante : S-Edit utilise l'API DOM pour le chargement et l'écriture de fichier XML alors que MIMOSA emploi l'API SAX. On est obligé de mettre en place une série de classe exploitant l'API SAX pour le chargement des formalismes S-Edit. En ce qui concerne l'écriture, la sauvegarde est morcelée dans chaque élément du formalisme MIMOSA à travers les méthodes **toXML()**.

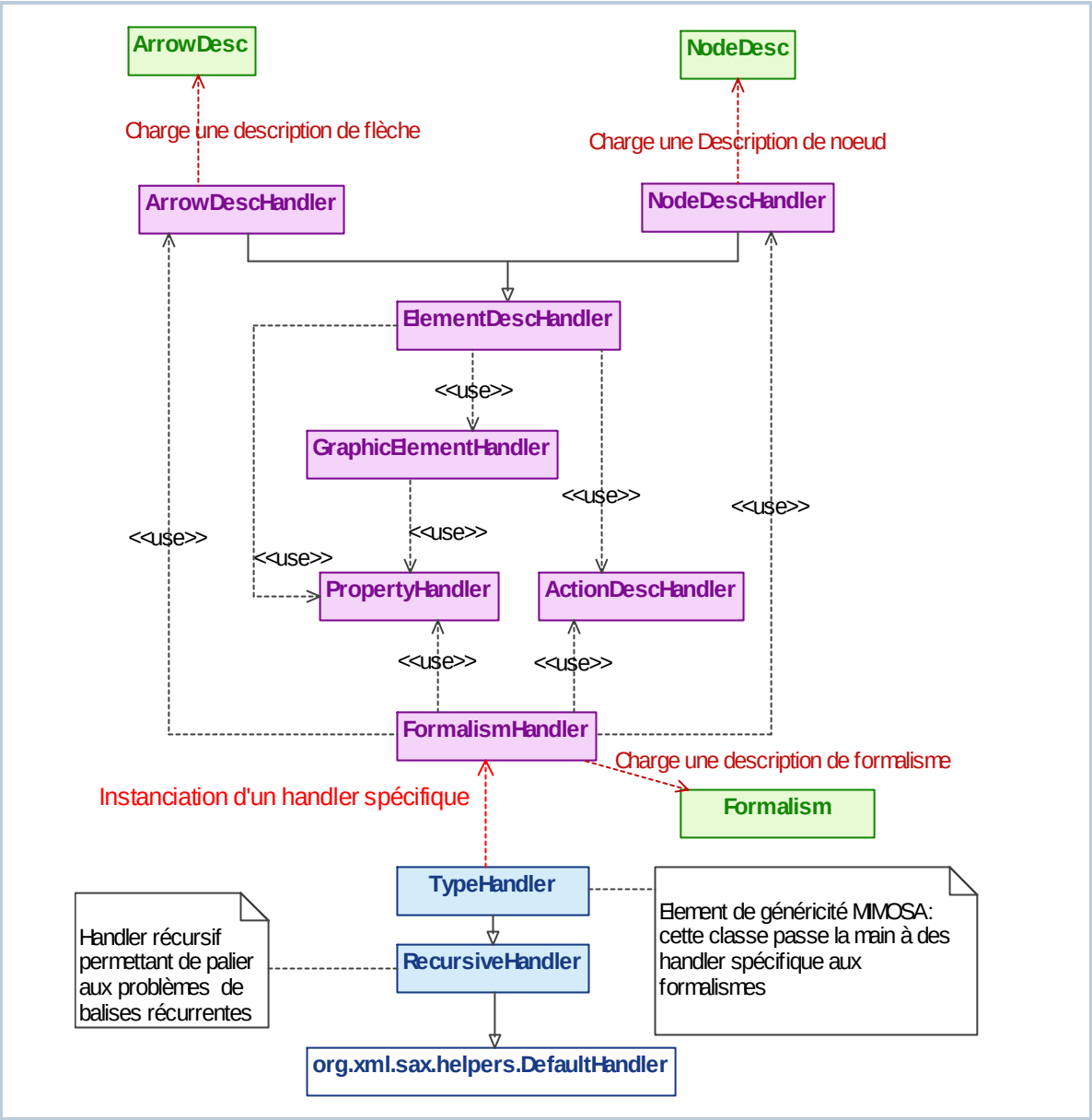


Figure 28. *Gestion de la sauvegarde à travers l'API SAX*

6. Bilan de l'assimilation par héritage

La technique d'assimilation par héritage a effectivement permis d'enrichir les composants MIMOSA de méta données propres à l'édition des modèles. Les modèles S-Edit étant des déclinaisons de la boite à outils S-Edit, la boite à outils S-Edit étant à présent une déclinaison de la boite à outils MIMOSA, alors tout les modèles S-Edit sont à présent des modèles MIMOSA. L'avantage de cette technique est qu'elle permet, outre la migration de la fonctionnalité d'édition, d'intégrer tous les formalismes et modèles définis à l'aide du système S-Edit. MIMOSA dispose donc à présent de notions lui permettant de décrire conceptuellement les modèles à base d'automates d'état transition, les schémas de circuits intégrés, les réseaux de pétri, les modèles à compartiments, etc.… L'avantage de cette technique est peut être aussi son plus gros inconvénient. La relation d'héritage telle qu'elle existe dans les langages de programmation induit un couplage très fort entre classes. Il en résulte que pour que les modèles MIMOSA puissent bénéficier de l'édition des modèles, ils doivent nécessairement être définis comme sous classes de la boite à outils S-Edit, laquelle rappelons le a été unifié à la boite à outils MIMOSA. Il s'en suit que si d'emblée les modèles de S-Edit sont inclus, les modèles MIMOSA sont exclus par la solution : ils doivent être réécris.

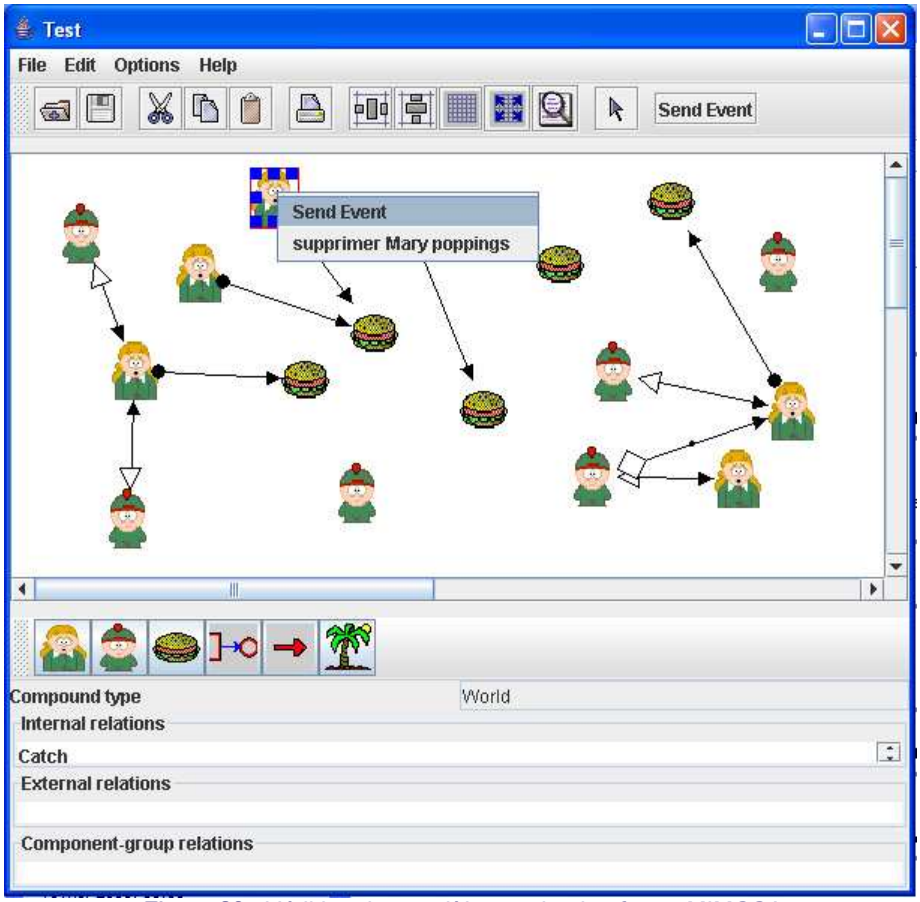


Figure 29. *L'édition des modèles sur la plateforme MIMOSA*

Or il ne faut pas perdre de vue que nous réalisons un couplage, un couplage est caractérisé par une certaine étique. Si le couplage est réalisé entre deux systèmes, il n'en demeure pas moins que le couplage ne se fait qu'au profit d'un système. Le couplage entre MIMOSA et S-Edit est fait au profit de MIMOSA. Les classes de la boite à outils de MIMOSA sont le support de la généricité de la plateforme, elles font partie de l'identité de la plateforme. Il est important de pouvoir dire de façon universelle que les modèles s'expriment à l'aide de Component, Compound et Relation. Certes les notions S-Edit peuvent être comparées aux notions de description du temps, des populations ou encore de l'espace. Sauf que tout modèle nécessite une construction donc une édition. L'usage des notions S-Edit va donc tendre à devenir une règle …

Résultat des jeux l'assimilation par délégation manque de flexibilité en ce sens qu'elle ne permet pas directement d'éditer des modèles issue de la déclinaison de la boite à outils MIMOSA, ce qui nous pousse à proposer une autre approche, une approche utilisant la délégation.

C.L'assimilation par délégation

1. Description

Depuis le début de la phase de conception de ce travail nous avons fais table rase des notions de description des modèles sur le plan dynamique. L'assimilation par délégation va justement nous permettre de traiter ce point. Nous avons constaté dans la partie analyse que les notions dynamiques proposées par MIMOSA sont identiques aux concepts du formalisme du diagramme d'état transition d'UML. Or le système S-Edit implémente le formalisme du diagramme d'état transition. Et naturellement on en vient à penser que l'on devrait pouvoir éditer à l'aide d'un diagramme S-Edit les notions dynamiques de MIMOSA. Cependant avec l'approche par héritage, on devrait alors dire que les classes qui implémentent les notions de nœuds et d'arcs doivent, en plus d'hériter des classes Component et Relation, hériter des classes qui implémentent les notions d'états et d'événements. Nous sommes dans un environnement Java, l'héritage multiple n'est pas permis. Il est nécessaire de le simuler, il est nécessaire de réaliser la propagation des propriétés manuellement, il est nécessaire de recourir à la délégation.

En plus de toutes ces considérations, il ne faut pas perdre de vue que l'édition graphique des modèles est un cas particulier de la visualisation des modèles. Or dans la visualisation des modèles dépendamment des besoins et dépendamment des modélisateurs un même modèle doit pouvoir être visualisé différemment. Dans notre cas un modélisateur à l'aise avec les formalismes UML pourrait souhaiter voir son modèle lors de la conception comme un ensemble d'objets du diagramme d'objets, un autre pourrait vouloir une visualisation plus imagé. Or si un couplage aussi fort que l'héritage a été établit entre les classes MIMOSA et leur visualisation sous forme de diagramme, il n'est pas possible de disposer de plusieurs visualisations schématiques d'un même modèle. Encore une fois il est nécessaire de recourir à la délégation.

2. Implémentation

Pour réaliser l'assimilation par délégation nous proposons deux approches. La première présente un certain manque de « propreté » mais a le bénéfice de conserver tout les formalismes définis sur S-Edit. La deuxième approche est parfaitement générique mais nous impose une réécriture totale de l'ensemble des formalismes S-Edit, pour peu qu'ils soient pertinents.

La première approche consiste à réécrire Les classes de la boite à outils S-Edit de sorte à encapsuler les classes de la boite à outils MIMOSA (Aussi bien les classes définissant

des notions statiques que des notions dynamiques). Le principal défaut de cette démarche c'est que la boite à outils S-Edit est modifiée. Mais dans une phase de test c'est l'approche la plus efficace. Mais de façon générale nous lui préférerons une approche plus élégante, en conformité avec le pattern décorateur, qui consiste à créer des sous classes des classes de la boite à outils S-Edit, de créer un quelque sorte un formalisme S-Edit. Ce sont les classes de ce formalismes qui vont encapsuler les classes MIMOSA.

Devant mettre en place un formalisme S-Edit, il est nécessaire de savoir que S-Edit ne prévoit pas l'extension des classes définissant les notions catégorielles, les types. A la place les notions catégorielles sont implémentées par les attributs et méthodes statiques des classes qui implémentent les notions individuelles. Au niveau de S-Edit se sont les notions catégorielles qui permettent de créer les notions individuelles. En effet les classes d'implémentation des notions catégorielles disposent d'une méthode de création d'instance (instance au niveau modélisation). Cette méthode effectue une instanciation (dans le sens programmation) de la classe qui implémente le concept individuel. En encapsulant un concept individuel dans un autre, il nécessaire que cette méthode instancie aussi le concept encapsulé. Cela implique que la méthode de création doit être réécrite, doit être redéfinie. Or pour redéfinir une méthode il n'existe que deux possibilités. La première consiste à réécrire les classes génériques de la boite à outils. C'est précisément ce que nous ne voulons pas. La deuxième possibilité consiste à mettre en place des sous classes redéfinissant cette méthode. Cela est faisable, mais dans l'approche par assimilation on est sous la législation de S-Edit et dans ce cas on irait à l'encontre de la philosophie de S-Edit en ce qui concerne les catégories. La seule solution qui nous reste c'est de mettre en place sur les classes d'instances un attribut statique qui va permettre d'encapsuler les types MIMOSA. Ainsi au lieu de redéfinir la méthode de création au niveau des catégories nous devrons plutôt redéfinir les constructeurs des instances de sorte à ce qu'ils fassent appel au type MIMOSA. Les types MIMOSA sont chargés de la création des instances.

Nous proposons donc les classes suivantes :

- **La classe ComponentNode** : cette classe hérite de **SNode** et agrége la classe MIMOSA **Component**. Cette classe dispose d'un attribut **componentType** statique publique pour encapsuler les types de composants.
- **La classe IntraCompoundRelationArrow** : cette classe hérite de **SArrow** et agrége la classe **IntraCompoundRelation**. Cette classe dispose d'un attribut

IntracompoundRelationType statique publique pour encapsuler les types de relations intra-composés.

- **La classe CompoundStructure** : cette classe hérite de **Structure** et agrège la classe MIMOSA **Compound**. Cette classe dispose d'un attribut **compoundType** statique publique pour encapsuler les types de composés.
- **La classe StateNode** : cette classe hérite de **SNode** et agrége les classes MIMOSA qui implémentent l'interface **State**.
- **La classe EventArrow** : cette classe hérite de **SArrow** et agrége la classe MIMOSA **SimEvent**. Cette classe dispose d'un attribut **eventType** statique publique pour encapsuler les types d'événements.

Le diagramme de classe suivant présente la structure de ce système :

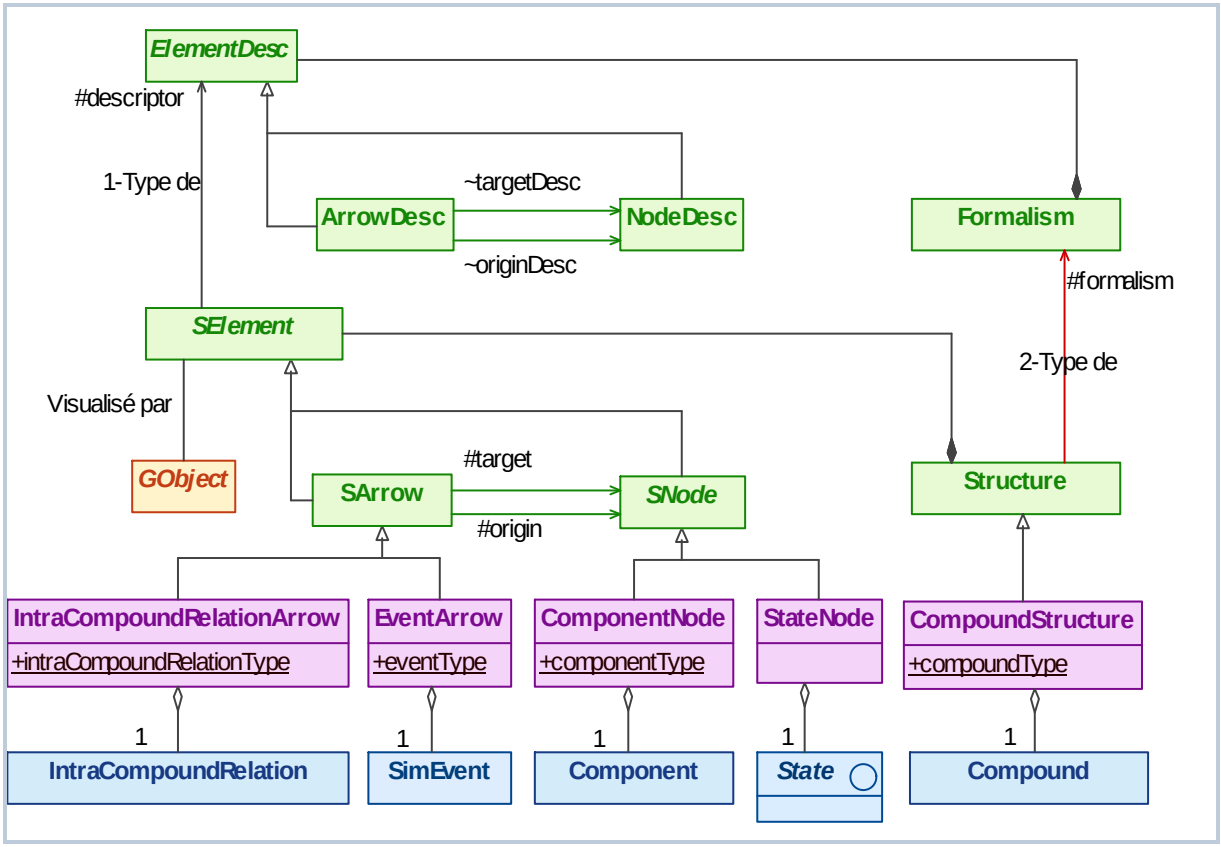


Figure 30. *Boites à outils : assimilation par délégation*

Classes MIMOSA
Classes S-Edit
Classes de couplage

3. Dépendances

Dans cette approche nous n'avons pas développé de formalisme MIMOSA particulier il nous faut cependant en définir un pour nous rattacher à la plateforme. Nous définirons donc :

- un composant **SEditStructureDiagram** pour représenter les diagrammes S-Edit de modélisation de la structure des modèles ;
- un composant **SEditDynamicDiagram** pour représenter les diagrammes S-Edit de modélisation de l'aspect dynamique des modèles ;
- un composé **SEditDiagrams** pour représenter un ensemble de diagrammes ;
- une relation inter-composés **StructureEdition** pour associer un modèle à un éditeur de structure ;
- une relation inter-composés **DynamicsEdition** pour éditer les comportements dans un modèle.

Ainsi que leurs types respectifs.

4. Bilan de l'assimilation par délégation

Premièrement il faut noter que l'assimilation par délégation n'a entraîné aucune modification de l'implémentation des deux systèmes. C'est ainsi l'approche par excellence pour un couplage. L'assimilation par délégation nous a permis de définir ce qu'il convient de qualifier de visualisation conceptuelle des modèles. En effet l'intégration de S-Edit dans cette approche s'est traduite par la création d'un monde de diagrammes, un modèle dont les composants sont des diagrammes. Or les diagrammes, offrent une vision schématique des modèles particulièrement adaptée à la conception. Tout modélisateur désirant construire un modèle, va établir une relation entre son modèle et l'univers des diagrammes. L'univers des diagrammes étant constitué d'un ensemble de diagrammes spécifiques, de formalismes de diagramme qu'il s'agira d'instancier et de mettre en relation avec un autre modèle. Cette approche à la fois élégante et très flexible, ne souffre que de défauts mineurs, des défauts d'ordre logistique car il sera nécessaire de développer à nouveau des formalismes S-Edit ayant pour base les classes que nous avons définies. L'intégration de S-Edit par délégation mets en exergue des possibilités, des fonctionnalités supplémentaires que S-Edit devrait implémenter dans le futur. Car si une vue conceptuelle a été mise en place, cela implique que des modèles déjà construits doivent être visualisables, leurs structures devraient pouvoir être révélées par mise en

relation avec le monde des diagrammes. Nous reviendrons sur ce point dans le chapitre suivant.

Chapitre V.
Conclusion et perspectives

A.Résumé

Au cours de cet exposé nous avons défini ce que devrait être l'édition de modèle sur une plateforme générique telle que MIMOSA. Nous avons vérifié que le système générique S-Edit qui nous était proposé par l'énoncé de notre sujet implémentait la définition que nous avions établie. Nous nous sommes ensuite intéressé au couplage des deux systèmes. Nous avons déterminé que le couplage devait se traduire par une fusion des éléments génériques de déclinaison des modèles : les boites à outils. Nous avons étudiés de façon comparative les boites à outils des deux systèmes. Nous avons conclu de cette étude que pour réaliser cette fusion nous avions le choix entre une assimilation statique et une assimilation dynamique. L'assimilation statique a été implémentée avec succès, cependant nous nous sommes rendu compte que l'assimilation dynamique est plus flexible et permet de définir la notion de visualisation conceptuelle.

B.Fonctionnalités supplémentaires

Nous proposons qu'en ce qui concerne la poursuite du projet MIMOSA, en ce qui concerne S-Edit que le couplage soit implémenté par une assimilation dynamique. L'assimilation dynamique permettant une vue conceptuelle des modèles nous proposons que soient mis en place des algorithmes de génération de diagrammes à partir de modèles déjà construits. Ces algorithmes permettront de révéler la structure des modèles, offrant ainsi une vue fonctionnelle des modèles. La réalisation de cette fonctionnalité devrait se traduire par la mise en place d'assistants d'importation de diagrammes (plus précisément d'instance de diagrammes).

Toujours en ce qui concerne S-Edit l'inspecteur de propriété pourrait être plus générique en intégrant les interfaces graphiques personnalisées d'édition associées aux concepts individuels de MIMOSA. Le concept de propriété peut être assez réducteur dés lors qu'il s'agit de l'édition de l'état des concepts. A l'opposé la plateforme MIMOSA devrait exploiter

de façon approfondie les possibilités utilitaires offertes par la notion de propriété. La notion de propriété permet d'accroître la généricité au niveau des interfaces utilisateurs. En effet une propriété est caractérisée par un type de donnée et un éditeur spécifique. Il serait intéressant de constituer une bibliothèque de types placés à un niveau d'abstraction supérieur associé à des contrôles utilisateurs spécifiques à la plateforme. On disposerait ainsi de types temps, température, vitesse associés à des jauges, des sliders. La mise en place de ces propriétés devrait permettre, de pair avec des conventions de codage, d'alléger significativement le travail des programmeurs de formalismes. Les conventions de codage permettront de générer les interfaces utilisateurs. Un stade d'atomicité plus élevé dans la généricité des interfaces graphiques serait ainsi atteint. Des outils génériques plus atomiques donc plus précis.

C.Développement de la plateforme

Le module de visualisation de la plateforme MIMOSA est en cours de développement. La présence d'un module de visualisation dans la plateforme MIMOSA va signifier que sera mis à disposition un modèle dont les composants sont des objets géométriques. Etant des composants MIMOSA, ces objets géométriques bénéficierons de la fonctionnalité de simulation de la plateforme, pour voir leur forme varier au fil du temps. L'intérêt du module de visualisation réside dans les relations entre la visualisation et les modèles visualisés. La modification d'un concept doit entraîner la modification de sa visualisation. D'un point de vue architectural nous proposons que la représentation graphique des graphes desservie par les sous classes de la classe S-Edit **GObject** soit un sous formalisme du formalisme de visualisation graphique. Ce n'est toutefois pas un impératif, car la variation d'une forme graphique est rarement significative dans le monde des diagrammes : une forme graphique est plutôt un symbole.

A travers l'assimilation par délégation nous avons mis en place un formalisme S-Edit, dans lequel nous nous sommes servis des éléments S-Edit pour encapsuler les éléments basiques des formalismes MIMOSA. Les composants ont été encapsulé de sorte que la création d'un nœud de graphe soit assimilée à la création d'un composant. Mais pourquoi se limiter à l'encapsulation des composants ? Pourquoi ne pas encapsuler des composés à la place, pourquoi ne pas développer un formalisme où les nœuds prendraient les responsabilités des composés et où les arcs prendraient les responsabilités des relation inter-composés. La résultante de cette démarche serait la possibilité de construire des systèmes complexes, d'éditer des multi modèles à l'aide de S-Edit. Dans ce cas de figure une interaction utilisateur comme le double clic sur un nœud permettrait d'ouvrir une fenêtre

d'édition de composés. Ainsi ce formalisme S-Edit pourrait permettre de basculer entre deux niveaux d'abstraction. On aurait ainsi une représentation imagée, concrète d'une approche holonique.

D.Conclusion

MIMOSA est un outil générique de modélisation dont l'implémentation de base correspond à la définition d'un système complexe, un système complexe qu'il s'agit de spécifier, de décrire. A travers son architecture MIMOSA introduit une philosophie de modélisation où l'élément central est la relation. S-Edit dispose d'une fonctionnalité de simulation des modèles, cependant nous n'avons pas cherché explicitement à intégrer cette fonctionnalité même si le couplage l'a introduit. Cela pour la simple raison que dans la philosophie MIMOSA, la simulation d'un modèle est une simple mise en relation du modèle avec un modèle du temps. Cette démarche aussi simple soit elle est redoutable de flexibilité. Disons par exemple que le modèle temporel soit inadéquat à la simulation précise d'un modèle X. Dans la philosophie MIMOSA on dit : « Je me souviens que les chercheurs du labo Y à Bamako ont défini une modélisation du temps qui pourrait nous convenir ; essayons mettons notre modèle en relation avec leur modèle du temps ». Cela implique qu'il n'existe pas une simulation mais autant de simulations que nécessaire. Tiens, nous voulons construire notre modèle, nous avons besoin de diagrammes conceptuels, et bien mettons notre modèle en relation avec le monde des diagrammes. Nous voulons le visualiser et bien mettons le en relation avec le monde des formes graphiques. Nous voulons le simuler, choisissons une modélisation du temps. La plateforme MIMOSA permet une véritable approche collaborative, une intégration harmonieuse des expériences, harmonieuse car permettant aux expériences, aux définitions divergentes de coexister sur la même plateforme. En ce sens, pour l'heure, à titre de contribution personnelle, nous pouvons dire que la plateforme réalise parfaitement les objectifs fédérateurs du projet MIMOSA.

Annexe A. Bibliographie

[1] Jean Pierre MULLER,
Mimosa : représentation et simulation des connaissances,
20 septembre 2004.

[2] Maturana, H.
Ontology of observing :
the biological foundations of self-consciousness
and the physical domain of existence. In American
society for Cybernetics Conference,
1988.

[3] Jacques Ferber,
Defining a new formalism,
www.lirmm.fr/~ferber/SEdit/

[4] J.-M. Legay,
L'expérience et le modèle, un discours sur la méthode,
INRA Editions, Paris, 1997.

[5] Wikipedia,
Extensible Markup Language (XML),
http://fr.wikipedia.org

[6] Pierre Alain Muller, Nathalie Gaertner,
Modélisation objet avec UML,
2000

[7] Deitel & Deitel,
Java Comment programmer – Quatrième édition,
2002

Annexe B. DTD des fichiers XML de description de formalismes S-Edit

```
<?xml encoding="US-ASCII"?>
<!-- This SEdit-Formalisms DTD (c) 1999 by MadKit/SEdit Development Team -->

<!-- Element descriptions -->

<!ELEMENT formalism (formalism-info,connector-types?,node-types,arrow-types?,action*)>
<!ELEMENT formalism-info (author+,doc?,icon?)>
<!ELEMENT java-method EMPTY>
<!ELEMENT scheme-function (#PCDATA)>
<!ELEMENT action (icon?,(java-method|scheme-function))>
<!ELEMENT property (#PCDATA)>

<!ELEMENT arrow-desc (icon?,graphic-element?,property*,action*)>
<!ELEMENT arrow-types (arrow-desc+)>

<!ELEMENT connector-desc (icon?,graphic-element?,property*)>
<!ELEMENT connector-types (connector-desc+)>

<!ELEMENT connector (property*)>
<!ELEMENT module (connector*,property*)>

<!ELEMENT icon EMPTY>
<!ELEMENT graphic-element (property*)>
<!ELEMENT node-desc (icon?,graphic-element?,module?,property*,action*)>
<!ELEMENT node-types (node-desc+)>

<!ELEMENT doc EMPTY>
<!ELEMENT author (#PCDATA)>

<!-- Attributes -->
<!ATTLIST icon
url CDATA #REQUIRED
>

<!ATTLIST java-method
name NMTOKEN #REQUIRED>
<!ATTLIST scheme-function
name NMTOKEN #REQUIRED>

<!ATTLIST action
name NMTOKEN #IMPLIED
description CDATA #REQUIRED
>

<!ATTLIST property
name NMTOKEN #REQUIRED>

<!ATTLIST arrow-desc
```

```
name ID #REQUIRED
description CDATA #IMPLIED
class NMTOKEN #IMPLIED
to IDREF #IMPLIED
from IDREF #IMPLIED
category NMTOKEN #IMPLIED
>

<!ATTLIST node-desc
name ID #REQUIRED
description CDATA #IMPLIED
class NMTOKEN #IMPLIED
category NMTOKEN #IMPLIED
>

<!ATTLIST connector-desc
name ID #REQUIRED
description CDATA #IMPLIED
class NMTOKEN #IMPLIED
mode (In | Out) #REQUIRED
category NMTOKEN #IMPLIED
>

<!ATTLIST module
type (Free | Fixed) #REQUIRED
layout (Auto | Justified | Manual) "Auto"
>

<!ATTLIST connector
type IDREF #REQUIRED
name NMTOKEN #IMPLIED
side (Left | Top | Right | Bottom) #IMPLIED
ratio CDATA #IMPLIED
>

<!ATTLIST graphic-element
class NMTOKEN #IMPLIED
>

<!ATTLIST doc
url CDATA #REQUIRED
>

<!ATTLIST formalism
name ID #REQUIRED
description CDATA #IMPLIED
class NMTOKEN #IMPLIED
>
```

Annexe C. Guide utilisateur de l'éditeur de modèles MIMOSA

Index